OWASP PROACTIVE CONTROLS FOR DEVELOPERS 2018 v 3.0

10 Critical Security Areas That Software Developers Must Be Aware Of

PROJECT LEADERS

KATY ANTON
JIM MANICO
JIM BIRD

1

OWASP PROACTIVE CONTROLS

DLA INŻYNIERÓW OPROGRAMOWANIA (DEWELOPERÓW)

2018 wersja 3.0

10 kluczowych obszarów zabezpieczeń, o których muszą pamiętać twórcy oprogramowania

Liderzy Projektu:
Katy Anton
Jim Manico
Jim Bird

2

Projekt OWASP podsumowujący dziesięć najlepszych rozwiązań w zakresie proaktywnej kontroli struktur zabezpieczających

Kilka słów na temat działalności organizacji OWASP

Open Web Application Security Project (OWASP) to charytatywna organizacja edukacyjna funkcjonująca na zasadach non-profit (501c3), której celem jest umożliwienie różnorodnym instytucjom projektowania, opracowywania i rozwijania, nabywania, obsługiwania oraz utrzymywania bezpiecznego oprogramowania. Wszelkie narzędzia, dokumenty, fora i publikowane przez OWASP materiały są bezpłatne i dostępne dla wszystkich zainteresowanych poprawą bezpieczeństwa aplikacji. Można je znaleźć na stronie www.owasp.org.

OWASP to zupełnie nowy rodzaj organizacji. Jako instytucja niezależna i wolna od komercyjnych nacisków przekazujemy całkowicie bezstronne, praktyczne i opłacalne informacje na temat bezpieczeństwa aplikacji.

Organizacja OWASP nie jest powiązana z żadną firmą technologiczną. Podobnie jak wiele innych projektów zajmujących się oprogramowaniem typu open source, OWASP tworzy i publikuje szereg różnorodnych materiałów na zasadach otwartości i szeroko zakrojonej współpracy. Fundacja OWASP jest organizacją non-profit, która zapewnia długofalowy sukces projektu.

3

Przedmowa

Oprogramowanie, które nie jest w pełni bezpieczne stanowi nie tylko bezpośrednie zagrożenie, lecz jest przede wszystkim czynnikiem osłabiającym szereg kluczowych infrastruktur na całym świecie, m.in. w obszarach finansów, ochrony zdrowia, energii i energetyki etc. Ponieważ globalna cyfrowa infrastruktura informacyjna staje się coraz bardziej złożona i wzajemnie powiązana, wszelkie problemy w zapewnieniu bezpieczeństwa aplikacji wzrastają wykładniczo. Nie można już dłużej akceptować istnienia względnie prostych problemów związanych z bezpieczeństwem, należy je bezwzględnie eliminować.

GŁÓWNE CELE PROJEKTU

Celem projektu OWASP Top 10 Proactive Controls (OPC) [*10 kluczowych obszarów zabezpieczeń*] jest podniesienie świadomości na temat bezpieczeństwa aplikacji poprzez opisanie najważniejszych obszarów, o których deweloperzy oprogramowania powinni wiedzieć i pamiętać. Zachęcamy tym samym do korzystania z dokumentu OWASP Proactive Controls, którego celem jest przybliżenie aktualnie dostępnych rozwiązań w zakresie proaktywnej kontroli struktur zabezpieczających. Dzięki nim programiści mogą uczyć się na błędach innych organizacji. Mamy nadzieję, że dokument OWASP Proactive Controls okaże się przydatny i będzie dużym wsparciem w tworzeniu bezpiecznego oprogramowania.

WEZWANIE DO DZIAŁANIA

W razie jakichkolwiek pytań, komentarzy, czy nowych pomysłów należy skontaktować się z liderami projektu OWASP Proactive Controls za pośrednictwem publicznie dostępnej listy e-mailowej lub pisząc prywatnie na adres jim@owasp.org.

PRAWA AUTORSKIE I LICENCJE

LIDERZY PROJEKTU

Katy Anton Jim Bird Jim Manico

WSPÓŁTWÓRCY

Chris Romeo Dan Anderson David Cybuck
Dave Ferguson Josh Grossman Osama Elnaggar
Colin Watson Rick Mitchell i wielu innych

Struktura dokumentu

Struktura niniejszego dokumentu ma charakter listy obszarów kontroli zabezpieczeń. Każdy z nich opisano w następujący sposób:

Charakterystyka

Szczegółowy opis obszaru wraz z przykładami dobrych praktyk.

Implementacja

Przykłady najlepszych sposobów zastosowania danego rozwiązania wraz ze szczegółowym opisem implementacji.

Luki w zabezpieczeniach (prewencja)

Lista luk i zagrożeń, których można uniknąć stosując dane rozwiązanie (vide: OWASP TOP 10 Risk, CWE, etc.)

Rekomendowane zasoby

Lista polecanych zasobów i materiałów do zapoznania się w następnej kolejności (np. OWASP Cheat sheet lub Security Hardening Guidelines - [Wytyczne dot. tworzenia protokołów zabezpieczeń wolnych od luk]

Narzędzia

Zestaw narzędzi/projektów służących do łatwego wprowadzania/integrowania kontroli bezpieczeństwa do/z istniejącym oprogramowaniem.

Wprowadzenie

OWASP Top Ten Proactive Controls 2018 to lista technik zabezpieczeń, które warto stosować w trakcie tworzenia każdego nowego projektu oprogramowania. Niniejszy dokument został opracowany z myślą o programistach, aby wspomóc ich w zakresie projektowania bezpiecznych rozwiązań.

Jednym z głównych celów tego dokumentu jest przedstawienie konkretnych praktycznych wskazówek, które mają pomóc programistom w tworzeniu bezpiecznego oprogramowania. Omawiane techniki powinny być stosowane proaktywnie na wczesnych etapach tworzenia i rozwoju oprogramowania, po to by zapewnić ich maksymalną skuteczność i bezpieczeństwo.

10 kluczowych obszarów zabezpieczeń

Lista została uporządkowana w oparciu o skalę ważności, której najistotniejszym elementem jest pozycja numer 1:

C1: Zdefiniuj wymagania bezpieczeństwa
C2: Wykorzystaj dostępne struktury bezpieczeństwa i biblioteki
C3: Zabezpiecz dostęp do baz danych
C4: Zastosuj kodowanie i escaping
C5: Sprawdzaj wszystkie dane wejściowe
C6: Zaimplementuj tożsamość cyfrową
C7: Wymuszaj kontrolę dostępu
C8: Chroń dane wszędzie
C9: Wprowadź rejestrowanie i monitorowanie bezpieczeństwa systemu
C10: Usuń wszelkie błędy i wyjątki

Jak powstała niniejsza lista

Lista ta została opracowana przez liderów projektu przy współpracy z grupą kilkunastu wolontariuszy. Dokument został następnie udostępniony na całym świecie, aby można było wziąć pod uwagę nawet anonimowe sugestie. Dzięki przyjęciu procedury otwartej na publiczną refleksję udało się wprowadzić cały szereg istotnych zmian oraz umieścić wartościowe informacje.

Docelowa grupa odbiorców

Niniejszy dokument sporządzono przede wszystkim na użytek programistów (deweloperów). Jednak menedżerowie ds. rozwoju oprogramowania, właściciele produktów, specjaliści ds. jakości i zarządzania, kierownicy programów projektowych oraz wszyscy zaangażowani w tworzenie oprogramowania mogą również korzystać z zawartych w nim informacji.

Jak korzystać z tego dokumentu

Dokument ma na celu dostarczenie podstawowej wiedzy na temat tworzenia bezpiecznego oprogramowania. Może on również stanowić solidny trzon informacji, szczególnie w trakcie szkoleń wprowadzających z zakresu bezpieczeństwa oprogramowania skierowanych do programistów. Omówione rozwiązania powinno się konsekwentnie stosować we wszystkich aplikacjach. Niemniej jednak niniejszy dokument należy raczej postrzegać jako punkt wyjścia do dalszych rozważań i procedur niż jako kompletny zestaw technik i zastosowań. Całościowy proces rozwoju bezpiecznego oprogramowania powinien uwzględniać szereg wymagań ujętych na przykład w standardach OWASP ASVS jak również działania programistyczne opisane szczegółowo w modelach dojrzałości takich jak OWASP SAMM i BSIMM.

Nawiązanie do projektu OWASP Top 10

Projekt OWASP Top 10 Proactive Controls jest zasadniczo podobny do OWASP Top 10, ale koncentruje się przede wszystkim na strategiach obrony przed naruszeniem danych i kontroli bezpieczeństwa, a nie na analizie i sterowaniu ryzykiem w systemach informatycznych. Każda technika lub proces kontrolny opisany w tym dokumencie będzie odnosić się do jednej lub więcej pozycji uwzględnionych w OWASP Top 10. Bliższe informacje na ten temat zostały zawarte na końcu każdego opisu poszczególnych strategii z listy.

C1: Zdefiniuj wymagania bezpieczeństwa

Charakterystyka

Wymagania dotyczące bezpieczeństwa określają jakie funkcjonalności systemu zabezpieczenia danych należy wprowadzić, aby sprawić, że oprogramowanie będzie posiadać konkretne właściwości w zakresie bezpieczeństwa. Wymogi te opierają się na standardach branżowych, obowiązujących przepisach prawa oraz historii wcześniejszych usterek i luk w oprogramowaniu. Zalecenia te definiują również nowe funkcje lub dodatki do istniejących funkcji w celu rozwiązania określonego problemu związanego z bezpieczeństwem lub wyeliminowania potencjalnej luki w zabezpieczeniach.

Ponadto stanowią one podstawę weryfikacji funkcji bezpieczeństwa dla aplikacji. Zamiast tworzyć indywidualne rozwiązania w zakresie bezpieczeństwa dla każdej aplikacji, zastosowanie standardowych wymagań i procedur pozwala programistom wielokrotnie korzystać z dostępnych definicji zabezpieczeń i wzorów najlepszych praktyk. Te same sprawdzone procedury zapewniają możliwość rozwiązania podobnych problemów związanych z bezpieczeństwem do tych, które wystąpiły już wcześniej. Wymagania opracowano po to, by zapobiegać powtarzaniu się tych samych awarii zabezpieczeń.

Standard OWASP ASVS

OWASP Application Security Verification Standard (ASVS) to katalog dostępnych wymagań dotyczących bezpieczeństwa oraz kryteriów weryfikacji. OWASP ASVS może stanowić dobre źródło informacji na temat szczegółowych wymagań bezpieczeństwa dla zespołów programistycznych.

Wymagania bezpieczeństwa są podzielone na różne segmenty w oparciu o wspólną funkcję bezpieczeństwa wyższego rzędu. Na przykład ASVS zawiera kategorie, takie jak uwierzytelnianie, kontrola dostępu, obsługa/rejestrowanie błędów i usługi sieciowe. Każda kategoria zawiera zbiór wymagań, które reprezentują najlepsze praktyki dla danej kategorii, opracowane jako sprawdzalne instrukcje.

Rozszerzanie wymagań o historie użytkowników i przypadki niewłaściwego użycia

Wymagania ASVS to podstawowe, dające się zweryfikować instrukcje, które można poszerzyć o historie użytkowników i przypadki niewłaściwego użycia. Zaletą historii użytkownika lub przypadków niewłaściwego użycia jest powiązanie aplikacji z dokładnie tym, co użytkownik lub osoba atakująca robi z systemem, a nie z opisem tego, co system oferuje użytkownikowi.

8

Oto przykład rozszerzenia zaleceń ujętych w ASVS 3.0.1. W sekcji "Wymagania dotyczące weryfikacji uwierzytelnienia" ASVS 3.0.1, w podpunkcie 2.19 skupiono się na domyślnych hasłach.

2.19 *Zweryfikuj czy nie wprowadzono haseł domyślnych w strukturze aplikacji lub w dowolnych komponentach używanych przez nią (takich jak "admin/password").*

Wymóg ten obejmuje zarówno działanie weryfikujące, czy w aplikacji nie zastosowano haseł domyślnych, jak również wskazanie iż w aplikacjach nie powinno się ich używać.

Historia użytkownika koncentruje się przede wszystkim na perspektywie samego użytkownika, administratora lub osoby atakującej system i opisuje daną funkcjonalność w oparciu o to, co użytkownik chce, aby system dla nich wykonał. Historia użytkownika ma postać "Jako użytkownik mogę zrobić x, y i z".

Jako użytkownik mogę wprowadzić swoją nazwę użytkownika i hasło, aby uzyskać dostęp do aplikacji.

Jako użytkownik mogę wprowadzić długie hasło, które może mieć maksymalnie 1023 znaki.

Kiedy historia użytkownika skupia się przede wszystkim na osobie atakującej i jej działaniach, określa się ją jako przypadek nadużycia.

Jako osoba atakująca mogę wprowadzić domyślną nazwę użytkownika i hasło, aby uzyskać dostęp.

Tak opracowany komunikat w historii użytkownika zawiera taką samą treść, co tradycyjny wymóg ASVS, dodatkowo jednak ujmuje informacje w kontekście użytkownika lub osoby atakującej, co sprawia, że dane zalecenie jest bardziej testowalne.

Wdrożenie/Implementacja

Pomyślne zastosowanie opracowanych zaleceń bezpieczeństwa obejmuje cztery kroki. Kolejne etapy procesu to odnajdywanie/selekcja, dokumentowanie, wdrażanie, a następnie potwierdzenie poprawnej implementacji wszelkich nowych elementów zabezpieczeń oraz funkcjonalności w aplikacji.

Odnajdywanie i selekcja

Proces rozpoczyna się od znalezienia i wyboru odpowiednich zaleceń dotyczących bezpieczeństwa. W tej fazie programista poznaje wymagania bezpieczeństwa ze standardowego źródła, na przykład ASVS, jak również dokonuje wyboru takiego wymogu, który warto lub należy uwzględnić dla danej wersji aplikacji. Celem takiego postępowania jest wybór łatwej do zarządzania liczby zaleceń dotyczących bezpieczeństwa dla danej wersji lub sprintu, a następnie kontynuowanie iteracji dla każdego sprintu, dodając z czasem więcej funkcjonalności zabezpieczających.

Badanie i dokumentacja

Na etapie badania i dokumentacji programista sprawdza istniejącą aplikację pod kątem nowego zestawu wymogów bezpieczeństwa, aby określić, czy aplikacja spełnia aktualne wymagania, czy też należy wprowadzić dodatkowe usprawnienia. Etap ten kończy się dokumentacją wyników przeglądu aplikacji.

Implementacja i testowanie

Po określeniu w jakim zakresie należy rozwinąć zabezpieczenia aplikacji programista powinien następnie zmodyfikować aplikację w taki sposób, aby albo dodać nową funkcjonalność, albo wyeliminować niezabezpieczoną opcję. W tej fazie programista najpierw określa w jaki sposób dostosować aplikację do aktualnych wymogów, a następnie wprowadza zmiany kodu, aby spełnić te wymagania. Należy również opracować przypadki testowe, aby potwierdzić wprowadzenie nowej funkcji lub wykazać istnienie wcześniej niezabezpieczonej opcji.

Luki w zabezpieczeniach (prewencja)

Wymagania bezpieczeństwa określają zakres funkcjonalności aplikacji w kontekście bezpieczeństwa. Im lepsze zabezpieczenia wbudowane od początku cyklu życia aplikacji tym lepsza możliwość zapobiegania wielu rodzajom luk w zabezpieczeniach.

Rekomendowane zasoby

- OWASP Application Security Verification Standard (Standardy OWASP dotyczące weryfikacji zabezpieczeń aplikacji - ASVS)
- OWASP Mobile Application Security Verification Standard (Standardy OWASP dotyczące weryfikacji zabezpieczeń aplikacji w urządzeniach mobilnych - MASVS)
- OWASP Top Ten

C2: Wykorzystaj dostępne struktury bezpieczeństwa i biblioteki

Charakterystyka

Bezpieczne biblioteki kodujące i platformy (frameworki) programowe z wbudowanym zabezpieczeniem pomagają programistom uchronić się przed zastosowaniem wadliwych rozwiązań w kontekście bezpieczeństwa. Programista piszący aplikację od podstaw może nie mieć wystarczającej wiedzy, czasu lub budżetu, aby prawidłowo zaimplementować lub utrzymać funkcje zabezpieczeń. Wykorzystanie dostępnych struktur bezpieczeństwa pomaga skuteczniej i lepiej realizować założone cele w zakresie bezpieczeństwa aplikacji.

Implementacja (najlepsze rozwiązania)

Włączając do swojego oprogramowania biblioteki lub frameworki należące do osób trzecich, należy wziąć pod uwagę następujące najlepsze praktyki:

1. Korzystaj z bibliotek i frameworków z zaufanych źródeł, które są aktywnie utrzymywane i powszechnie używane przez wiele aplikacji.
2. Utwórz i utrzymuj katalog inwentaryzacji wszystkich bibliotek należących do stron trzecich.
3. Proaktywnie aktualizuj biblioteki i komponenty. Używaj narzędzia, takiego jak OWASP Dependency Check czy Retire.JS, aby zidentyfikować zakres zależności projektu i sprawdzić, czy znane są publicznie jakiekolwiek luki w zabezpieczeniach kodów należących do stron trzecich.
4. Zmniejsz powierzchnię ataku, obudowując bibliotekę i pokazując tylko konkretne, wymagane zachowanie w twoim oprogramowaniu.

Luki w zabezpieczeniach (prewencja)

Bezpieczne frameworki i biblioteki mogą pomóc w zapobieganiu wielu rodzajom luk w aplikacjach internetowych. Niezwykle ważne jest aktualizowanie tych frameworków i bibliotek, co zostało szczegółowo opisane w dokumencie: korzystanie z komponentów o znanych podatnościach na zagrożenia - lista Top Ten 2017.

Narzędzia

- OWASP Dependency Check (Kontrola zależności opracowana przez OWASP) - identyfikuje zależności projektu i sprawdza publicznie ujawnione luki w zabezpieczeniach

- Skaner Retire.JS dla bibliotek JavaScript

C3: Zabezpiecz dostęp do baz danych

Charakterystyka

W tej sekcji opisano bezpieczny dostęp do różnych miejsc przechowywania danych, w tym zarówno relacyjnych baz danych, jak i baz danych NoSQL. Główne kwestie, które należy rozważyć to:

1. Bezpieczne zapytania
2. Bezpieczna konfiguracja
3. Bezpieczne uwierzytelnianie
4. Bezpieczna komunikacja

Bezpieczne zapytania

SQL Injection to metoda ataku komputerowego, która może mieć miejsce gdy niezaufane dane będą dynamicznie dodawane przez użytkownika do zapytania SQL w sposób niezabezpieczony, często poprzez podstawowe połączenie ciągów. SQL Injection stanowi jedno z największych zagrożeń z punktu widzenia bezpieczeństwa aplikacji. SQL Injection jest łatwy w użyciu i może doprowadzić do kradzieży, wyczyszczenia lub modyfikacji całej bazy danych. Aplikacja może być nawet używana do uruchamiania niebezpiecznych poleceń przeciwko systemowi operacyjnemu obsługującemu bazę danych, co daje osobie atakującej punkt oparcia w sieci.

Aby zmniejszyć zagrożenie SQL Injection, powinno się wprowadzić rozwiązania zapobiegające interpretacji wszelkich niezaufanych danych wejściowych jako części polecenia SQL. Najlepszym rozwiązaniem będzie tutaj zastosowanie techniki programowania znanej jako „parametryzacja zapytania". Takie rozwiązanie należy wprowadzić do SQL, OQL, a także uwzględnić w trakcie tworzenia procedur składowanych.

Dobrą listę przykładów parametryzacji zapytań w ASP, ColdFusion, C #, Delphi, .NET, Go, Java, Perl, PHP, PL/SQL, PostgreSQL, Pythonie, R, Ruby i Scheme można znaleźć na stronie http://bobby- tables.com oraz w OWASP Cheat Sheet on Query Parameterization (Ściągawka OWASP w zakresie parametryzacji zapytań).

Uwaga dotycząca parametryzacji zapytań

Niektórych lokalizacji w zapytaniu do bazy danych nie można parametryzować. Lokalizacje te mogą się różnić w zależności od dostawcy baz danych. Konfrontując parametry zapytania, których nie da się powiązać z zapytaniami sparametryzowanymi, należy bardzo dokładnie sprawdzić zgodność lub poddać je zabiegowi escapingu. Ponadto, chociaż użycie sparametryzowanych zapytań ma w dużej mierze pozytywny wpływ na ogólną wydajność systemu, niektóre sparametryzowane zapytania w konkretnych implementacjach bazy danych będą wpływać na nią negatywnie. Należy zatem pamiętać o tym, by testować zapytania pod kątem wydajności; szczególnie dotyczy to złożonych zapytań/kwerend o rozległej klauzuli podobnej lub z możliwością wyszukiwania tekstu.

12

Bezpieczna konfiguracja

Niestety systemy zarządzania bazami danych nie zawsze są dostarczane w konfiguracji „domyślnie bezpiecznej”. Należy zatem zadbać o to, aby zabezpieczenia dostępne z poziomu Systemu Zarządzania Bazą Danych (DBMS) i platformy hostingowej były włączone i odpowiednio skonfigurowane. Dostępne są standardy, przewodniki i testy porównawcze dla większości popularnych systemów DBMS.

Bezpieczne uwierzytelnianie

Cały dostęp do bazy danych powinien być odpowiednio uwierzytelniony. Uwierzytelnienie dla DBMS powinno być wykonane w bezpieczny sposób i powinno odbywać się wyłącznie za pomocą bezpiecznego kanału. Poświadczenia muszą być odpowiednio zabezpieczone i dostępne do użycia.

Bezpieczna komunikacja

Większość systemów zarządzania bazą danych (DBMS) obsługuje szereg metod komunikacji (usługi, interfejsy API itp.) zarówno bezpiecznych (uwierzytelnione, szyfrowane) jak i niezabezpieczonych (nieuwierzytelnione lub niezaszyfrowane). Dobrą praktyką jest używanie tylko bezpiecznych opcji komunikacji w ramach *Protect Data Everywhere* (chroń dane wszędzie).

Luki w zabezpieczeniach (prewencja)

- OWASP Top 10 2017- A1: Injection
- OWASP Mobile Top 10 2014-M1 Weak Server Side Controls

Rekomendowane zasoby

- OWASP Cheat Sheet: Query Parameterization (Ściągawka OWASP nt. parametryzacji zapytań)
- Bobby Tables: A guide to preventing SQL Injection (Przewodnik jak zapobiegać SQL Injection)
- CIS Database Hardening Standards (Standardy „hartowania” baz danych w CIS)

13

C4: Zastosuj kodowanie i escaping

Charakterystyka

Kodowanie i escaping to techniki obronne mające na celu powstrzymanie ataków typu Injection. Kodowanie (potocznie zwane „Kodowaniem Wyjściowym") polega na tłumaczeniu znaków specjalnych na inną, ale równoważną formę, która nie jest już niebezpieczna w tłumaczeniu docelowym, na przykład tłumaczenie znaku „**<**" na ciąg znaków **<** podczas pisania na stronie HTML. Escaping polega na dodaniu znaku specjalnego przed znakiem/ciągiem znaków, aby uniknąć błędnej interpretacji, na przykład dodanie znaku „****" przed znakiem „**"**"(cudzysłów), tak aby całość była interpretowana jako tekst, a nie jako znak zamykający ciąg.

Kodowanie wyjściowe najlepiej stosować **tuż przed** przekazaniem treści do docelowego interpretera. Jeśli w trakcie przetwarzania żądania ten rodzaj zabezpieczenia zostanie zastosowany zbyt wcześnie, to kodowanie lub escaping może ostatecznie zakłócić korzystanie z treści w innych częściach programu. Na przykład, jeśli HTML zawiera treść escape przed zapisaniem danych w bazie, a interfejs użytkownika automatycznie zastosuje ponowny escaping, wówczas zawartość nie będzie wyświetlać się poprawnie z powodu podwójnego escapingu.

Kontekstowe kodowanie wyjściowe

Kontekstowe kodowanie wyjściowe jest podstawową techniką bezpiecznego programowania potrzebną do zablokowania próby ataku typu XSS. Taki rodzaj zabezpieczenia wprowadza się niejako *na wyjściu* czyli na ostatnim etapie tworzenia interfejsu użytkownika tuż zanim niezaufane dane zostaną dynamicznie dodane do HTML. Rodzaj kodowania będzie zależał od lokalizacji (lub kontekstu) w dokumencie, w którym są wyświetlane lub przechowywane dane. Główne rodzaje kodowania, których można używać do tworzenia bezpiecznych interfejsów użytkownika to: kodowanie encji HTML, kodowanie atrybutów HTML, kodowanie JavaScript i kodowanie adresów URL.

Przykłady kodowania w Javie

Przykłady enkoderów Java OWASP zapewniających kontekstowe kodowanie wyjściowe można znaleźć w: OWASP Java Encoder Project Examples.

Przykłady kodowania w .NET

Począwszy od wersji .NET 4.5, biblioteka rozwiązań zapobiegających tworzeniu skryptów międzyserwisowych (Anti-Cross Site Scripting) stanowi integralną część struktury, ale nie jest domyślnie włączona. Można na przykład wybrać AntiXssEncoder z tej biblioteki jako domyślny dla całej aplikacji przy użyciu ustawień web.conf. Po wprowadzeniu takiego ustawienia należy jednak kontekstowo kodować wszelkie dane wyjściowe - innymi słowy zadbać o to, by używać właściwej funkcji z biblioteki AntiXSSEncoder dla odpowiedniej lokalizacji danych w dokumencie.

Przykłady kodowania w PHP

Zend Framework 2

W Zend Framework 2 (ZF2) Zend\Escaper może być używany do kodowania danych wyjściowych. Aby zapoznać się z przykładami kodowania kontekstowego warto zajrzeć do: Context-specific escaping with zend-escaper.

Inne typy kodowania i obrony przed atakiem SQL Injection

Kodowanie/Escaping może być użyte do neutralizowania treści dla uniknięcia innych form ataku typu Injection. Na przykład możliwe jest zneutralizowanie pewnych specjalnych znaków meta podczas dodawania danych wejściowych do polecenia systemu operacyjnego. Nazywa się to „ucieczką/escapingiem z systemu operacyjnego" (OS command escaping) lub „shell escaping" etc. Taki sposób obrony przed atakiem może zostać wykorzystany do zatrzymania luki "Command Injection". Istnieją też inne formy escapingu, które można wykorzystać do zablokowania ataku typu Injection, takie jak escaping atrybutu XML, który powstrzymuje przed różnymi formami ataku typu Injection ze ścieżki XML i XML, a także tzw. „LDAP distinguished name escaping", które może być użyte do zatrzymania różnego rodzaju ataków typu LDAP Injection.

Kodowanie znaków i kanonikalizacja

Kodowanie Unicode to metoda przechowywania znaków wielobajtowych. Wszędzie tam, gdzie dane wejściowe są dozwolone, można je wprowadzać przy użyciu Unicode i w ten sposób ukryć złośliwy kod lub też umożliwić różnego rodzaju ataki. RFC 2279 przytacza wiele sposobów kodowania tekstu.

Kanonikalizacja jest natomiast metodą, w której systemy konwertują dane do postaci prostej lub standardowej. Aplikacje internetowe często wykorzystują kanonikalizację znaków, po to by wszystkie przechowywane lub wyświetlane dane były tego samego typu.

Aby w pełni zabezpieczyć się przed atakami z wykorzystaniem kanonikalizacji, należy tak zaprojektować aplikację, by była ona bezpieczna nawet po wprowadzeniu zniekształconego kodu Unicode lub innych zniekształconych znaków.

Luki w zabezpieczeniach (prewencja)

- OWASP Top 10 2017 - A1: Injection
- OWASP Top 10 2017 - A7: Cross Site Scripting (XSS)

- OWASP Mobile_Top_10_2014-M7 Client Side Injection

Rekomendowane zasoby

- XSS - Informacje ogólne
- OWASP Cheat Sheet: XSS Prevention - Zatrzymywanie ataków typu XSS w aplikacji internetowej
- OWASP Cheat Sheet: DOM based XSS Prevention
- OWASP Cheat Sheet: Injection Prevention

Narzędzia

- OWASP Java Encoder Project
- AntiXSSEncoder
- Zend\Escaper - przykłady kodowania kontekstowego

16

C5: Sprawdzaj wszystkie dane wejściowe

Charakterystyka

Sprawdzanie danych wejściowych jest techniką programistyczną, która zapewnia że tylko prawidłowo sformatowane dane mogą wejść do komponentu systemu oprogramowania.

Analiza syntaktyczna i semantyczna

Dobrze zaprojektowana aplikacja powinna być w stanie sprawdzić, czy dane są poprawne pod względem składniowym i semantycznym (dokładnie w tej kolejności) jeszcze przed użyciem ich w jakikolwiek sposób (w tym wyświetleniem go użytkownikowi).

Analiza syntaktyczna oznacza sprawdzenie czy dane wejściowe mają oczekiwaną formę. Na przykład aplikacja może umożliwić użytkownikowi wybranie czterocyfrowego „identyfikatora konta" w celu wykonania jakiejś operacji. Aplikacja powinna również zakładać, że użytkownik może wprowadzać dane typu SQL Injection i powinna sprawdzić, czy dane wprowadzane przez użytkownika są dokładnie czterocyfrowej długości i czy faktycznie składają się wyłącznie z liczb (oprócz wprowadzenia prawidłowej parametryzacji zapytań).

Analiza semantyczna polega na sprawdzeniu i akceptowaniu wyłącznie takich danych wejściowych, które mieszczą się w dopuszczalnym zakresie dla danej aplikacji i jej kontekstu. Na przykład data rozpoczęcia musi być wcześniejsza niż data zakończenia przy wybieraniu zakresów dat.

Biała lista kontra czarna lista

Istnieją dwa ogólnie przyjęte podejścia do sprawdzania poprawności składni danych wejściowych. Są one powszechnie znane jako czarna lista (blacklisting) i biała lista (whitelisting):

Sprawdzenie przy użyciu czarnej listy (blacklisting/blacklist validation) polega na ustaleniu czy pewne konkretne dane wejściowe nie zawierają znanej już „złośliwej" treści. Na przykład aplikacja internetowa może blokować dane wejściowe zawierające dokładny tekst <**SCRIPT**>, aby pomóc w zapobieganiu atakom typu XSS. Zabezpieczenie to można jednak obejść stosując znacznik skryptu pisany małymi literami lub o mieszanej wielkości liter.

Sprawdzenie przy użyciu białej listy (whitelisting/whitelist validation) ma na celu ustalenie czy pewne konkretne dane wejściowe pasują do znanego zestawu „dobrych (niezłośliwych)" reguł. Na przykład zasada sprawdzania przy użyciu białej listy w obrębie USA byłaby dwuliterowym kodem, który odpowiada tylko jednemu ze Stanów.

W procesie tworzenia bezpiecznego oprogramowania wbudowanie opcji sprawdzania danych przy użyciu białej listy zaleca się jako zabezpieczenie minimum. Czarna lista jest podatna na błędy i można ją skutecznie ominąć za pomocą różnych dostępnych rozwiązań, sama może również stanowić potencjalne niebezpieczeństwo, gdy jest zależna „od siebie".

Mimo że często można ją obejść, czarna lista zdecydowanie ułatwia wykrycie oczywistych prób ataków. Zatem, o ile zastosowanie białej listy pomaga ograniczyć powierzchnię ataku, zapewniając, że wejściowe dane będą prawidłowe pod względem składni i semantyki, czarna lista znacznie ułatwia ich wykrywanie i może też potencjalnie powstrzymywać oczywiste próby ataków.

Sprawdzanie danych wejściowych po stronie klienta i po stronie serwera

Sprawdzanie poprawności danych wejściowych powinno zawsze odbywać się po stronie serwera w celu zapewnienia jak największego stopnia bezpieczeństwa. O ile sprawdzanie poprawności danych ze strony klienta może być przydatne zarówno pod względem funkcjonalnym, jak i w kontekście niektórych zabezpieczeń, zbyt łatwo można je jednak ominąć. Dlatego też tak istotne, wręcz fundamentalne dla bezpieczeństwa całej aplikacji, jest sprawdzanie danych ze strony serwera. Na przykład sprawdzanie poprawności danych wejściowych poprzez JavaScript może ostrzec użytkownika, że dane pole powinno składać się z liczb, ale dopiero aplikacja po stronie serwera ma za zadanie potwierdzić, że przesłane dane faktycznie zawierają wyłącznie cyfry w odpowiednim zakresie liczbowym dla danej cechy.

Wyrażenia regularne

Wyrażenia regularne to kolejny sposób sprawdzenia, czy dane pasują do określonego wzorca. Zacznijmy od prostego przykładu.

Następujące wyrażenie regularne służy do definiowania reguły białej listy, służącej do sprawdzania nazw użytkownika.

^[a-z0-9_]{3,16}$

Dopuszcza ono użycie wyłącznie małych liter, cyfr oraz znaku podkreślenia. Nazwa użytkownika jest również ograniczona do 3 i 16 znaków.

Uwaga: możliwość odmowy usługi

Podczas tworzenia wyrażeń regularnych należy zachować ostrożność. Źle zaprojektowane wyrażenia mogą spowodować zaistnienie warunków odmowy usługi (inaczej ReDoS). Dostępnych jest jednak szereg narzędzi testujących, aby zweryfikować czy wprowadzone wyrażenia regularne nie są podatne na ReDoS.

Uwaga: złożoność

Wprowadzenie wyrażeń regularnych jest jednym z wielu sposobów na sprawdzenie danych wejściowych. Dla niektórych programistów stosowanie ich może być jednak trudne w utrzymaniu lub nawet w zrozumieniu. Dobrą alternatywą jest zaprogramowanie metod sprawdzania (autorskim kodem). Takie rozwiązanie może być łatwiejsze do utrzymania dla niektórych programistów.

Granice możliwości sprawdzania danych wejściowych

Weryfikacja danych wejściowych nie zawsze sprawia, że dane są „bezpieczne”, ponieważ pewne formy złożonych danych wejściowych mogą zostać „pozytywnie zweryfikowane” chociaż nadal będą niebezpieczne. Na przykład poprawny adres e-mail może zawierać atak typu SQL Injection lub prawidłowy adres URL może zawierać próbę ataku typu Cross Site Scripting. Dlatego też oprócz sprawdzenia danych wejściowych powinno się zawsze stosować dodatkowe zabezpieczenia takie jak parametryzacja zapytań lub escaping.

Wyzwania jakim trzeba sprostać w trakcie sprawdzania serializowanych danych

Niektóre formy danych wejściowych są tak złożone, że ich sprawdzenie może jedynie minimalnie chronić aplikację. Na przykład wyjątkowo niebezpieczna jest deserializacja niezaufanych danych lub danych, którymi może manipulować osoba atakująca. Jedynym bezpiecznym wzorcem postępowania w tej sytuacji jest wbudowanie do systemu blokady akceptowania serializowanych obiektów pochodzących z niezaufanych źródeł lub ustawienie możliwości deserializacji wyłącznie prostych typów danych. Należy unikać przetwarzania

serializowanych formatów danych oraz, jeśli to tylko możliwe, korzystać z formatów, które łatwiej chronić czyli takich jak JSON.

Jeśli nie jest to możliwe, należy rozważyć przetwarzanie serializowanych danych za pomocą szeregu mechanizmów sprawdzania. Można na przykład:

- Wdrożyć sprawdzanie integralności lub szyfrowanie serializowanych obiektów, aby zapobiec nieupoważnionemu tworzeniu obiektów lub manipulowaniu danymi.

- Wymuszać ścisłe ograniczenia typów danych podczas deserializacji jeszcze przed utworzeniem obiektów; zazwyczaj kod oczekuje zestawu możliwych do zdefiniowania klas. Znane są już jednak sposoby obejścia takiego rozwiązania.

- Izolować kod, który deserializuje dane, tak aby działał jedynie w środowiskach o bardzo niskich uprawnieniach, takich jak tymczasowe kontenery.

- Prowadzić zapis wyjątków i nieudanych deserializacji, na przykład wtedy gdy wprowadzony typ danych nie jest oczekiwany lub gdy deserializacja generuje wyjątki.

- Ograniczyć lub monitorować przychodzące i wychodzące połączenia sieciowe z kontenerów lub serwerów, które deserializują dane.

- Monitorować deserializację, wysyłając ostrzeżenie jeśli konkretny użytkownik często deserializuje dane.

19

Nieoczekiwane dane wejściowe użytkownika (wykorzystanie luki Mass Assignment)

Niektóre struktury (frameworki) obsługują automatyczne powiązanie parametrów żądań HTTP z obiektami po stronie serwera używanymi przez aplikację. Funkcja automatycznego wiązania umożliwia osobie atakującej aktualizowanie takich obiektów po stronie serwera, które nie miały być modyfikowane. Za pomocą tej funkcji osoba atakująca może na przykład zmodyfikować swój poziom kontroli dostępu lub ominąć warstwę logiki biznesowej aplikacji.

Ten rodzaj ataku ma wiele różnych nazw np.: mass assignment, autobinding czy object injection.

Prosty przykład: jeśli obiekt użytkownika ma uprawnienia do pola, które określa poziom uprawnień użytkownika w aplikacji, złośliwy użytkownik może szukać stron, na których dane użytkownika są modyfikowane, i dodawać uprawnienia=admin (privilege=admin) do wysyłanych parametrów HTTP. Jeśli automatyczne wiązanie jest włączone w sposób niezabezpieczony, obiekt po stronie serwera reprezentujący użytkownika zostanie wtedy odpowiednio zmodyfikowany.

W tym celu można zastosować dwa podejścia. Można:

- Unikać bezpośredniego wiązania wprowadzonych danych i korzystać z obiektów przenoszenia danych (DTO - Data Transfer Objects).

- Włączyć automatyczne wiązanie, ale skonfigurować reguły białej listy dla każdej strony lub funkcji, tak aby określić, które pola mogą być automatycznie powiązane.

Więcej przykładów można znaleźć w OWASP Mass Assignment Cheat Sheet (Ściągawka OWASP nt. luki Mass Assignment).

Sprawdzanie i 'oczyszczanie' HTML'u

Rozważmy teraz aplikację, która musi akceptować HTML od użytkowników (poprzez edytor WYSIWYG, który ukazuje treść jako HTML lub funkcje, które bezpośrednio akceptują HTML na wejściu). W tej sytuacji ani sprawdzanie, ani escaping nie będą pomocne gdyż:

- Wyrażenia regularne nie są wystarczająco ekspresyjne, aby zrozumieć złożoność HTML5.

- Kodowanie lub escaping HTML'u nie spełni swojego zadania; spowoduje tylko, że HTML nie będzie prawidłowo renderowany.

Dlatego też w takiej sytuacji potrzebna jest biblioteka, która potrafi przetworzyć i „oczyścić" (usunąć) tekst w formacie HTML. Więcej informacji na ten temat można znaleźć w XSS Prevention Cheat Sheet on HTML Sanitization (Ściągawka nt. sposobów zapobiegania atakom typu XSS)

Funkcja sprawdzania danych wejściowych dostępna w bibliotekach i frameworkach

Wszystkie języki programowania i większość frameworków udostępnia swoje biblioteki do sprawdzania danych wejściowych lub funkcje, które można w tym celu użyć. Biblioteki zazwyczaj obejmują najbardziej podstawowe typy danych, wymagania dotyczące długości ciągu znaków, zakresy całkowite, kontrole typu „ma wartość zerową" etc. Wiele bibliotek i frameworków służących do sprawdzania danych wejściowych umożliwia również definiowanie własnego wyrażenia regularnego lub logiki dla niestandardowego sprawdzania poprawności w taki sposób, aby programista mógł wykorzystać tę funkcjonalność w całej aplikacji. Przykłady funkcjonalności sprawdzania danych obejmują filtrowanie PHP lub tzw. Hibernate Validator dla Javy. Przykłady sposobów „oczyszczania" HTML'u obejmują metodę Ruby on Rails, OWASP Java HTML Sanitizer lub DOMPurify.

Luki w zabezpieczeniach (prewencja)

- Sprawdzanie poprawności danych wejściowych zmniejsza obszar ataków aplikacji i może czasami utrudniać przeprowadzenie takiego ataku.

- Sprawdzanie poprawności danych wejściowych to rozwiązanie zapewniające bezpieczeństwo określonych rodzajów danych, specyficznych dla niektórych ataków, dlatego też nie może być rzetelnie stosowane jako ogólna reguła zabezpieczeń.

- Sprawdzanie danych wejściowych nie powinno być używane jako podstawowa metoda zapobiegania atakom typu XSS, SQL Injection i innym.

Rekomendowane zasoby

- OWASP Cheat Sheet: Input Validation (Ściągawka OWASP nt. sprawdzania danych wejściowych)
- OWASP Cheat Sheet: iOS - Security Decisions via Untrusted Inputs (Ściągawka OWASP nt. iOS - decyzje związane z zabezpieczeniem aplikacji w obliczu niezaufanych danych wejściowych)
- OWASP Testing Guide: Testing for Input Validation (Przewodnik OWASP: testowanie aplikacji pod kątem sprawdzania danych wejściowych)

Narzędzia

- OWASP Java HTML Sanitizer Project
- Java JSR-303/JSR-349 Bean Validation
- Java Hibernate Validator
- JEP-290 Filter Incoming Serialization Data
- Apache Commons Validator
- PHP’s filter functions

C6: Zaimplementuj tożsamość cyfrową

Charakterystyka

Tożsamość cyfrowa to unikalny zestaw danych umożliwiający identyfikację użytkownika (lub innego podmiotu), biorącego udział w transakcji online. W tym celu stosuje się uwierzytelnienie, aby sprawdzić czy dana osoba lub podmiot jest tym, za kogo się podaje. Z kolei zarządzanie sesją to proces, w którym serwer utrzymuje stan uwierzytelnienia użytkowników, dzięki czemu użytkownik może nadal korzystać z systemu bez konieczności ponownego uwierzytelniania. Publikacja NIST 800-63B: Wytyczne dotyczące tożsamości cyfrowej (uwierzytelnianie i zarządzanie cyklem życia aplikacji) [NIST Special Publication 800-63B: Digital Identity Guidelines (Authentication and Lifecycle Management)] zawiera praktyczne wskazówki dotyczące wdrażania tożsamości cyfrowej, uwierzytelniania i kontroli zarządzania sesją.

Poniżej znajdują się zalecenia dotyczące bezpiecznej implementacji procesu uwierzytelniania.

Poziomy uwierzytelniania

NIST 800-63b opisuje trzy poziomy uwierzytelniania użytkownika, po angielsku: Authentication Assurance Level (AAL). Poziom AAL 1 jest zarezerwowany dla aplikacji o niższym ryzyku, które nie zawierają danych osobowych ani innych prywatnych danych. Na poziomie AAL 1 wymagane jest tylko jednoetapowe uwierzytelnienie, zwykle za pomocą hasła.

Poziom 1: hasła

Hasła są naprawdę bardzo ważne. Polityka bezpieczeństwa i zarządzania danymi jest niezmiernie istotna. Ponadto należy przechowywać wszystkie dane w sposób bezpieczny, czasami zezwalając użytkownikom na ich usuwanie.

Wymagania dotyczące hasła

Hasła powinny spełniać przynajmniej następujące wymagania:

- powinny mieć co najmniej 8 znaków o ile zastosowano również uwierzytelnianie wieloskładnikowe (MFA) i inne kontrole bezpieczeństwa. Jeśli wprowadzenie MFA nie jest możliwe, należy zwiększyć długość hasła do co najmniej 10 znaków

- wszystkie znaki drukowane ASCII oraz znak spacji powinny być dopuszczalne w trybie zapamiętywania (memorized secrets)

- należy zachęcać użytkowników do stosowania długich haseł i fraz

- należy usunąć wymagania dotyczące złożoności hasła, ponieważ wykazano, że skuteczność takiego rozwiązania jest bardzo ograniczona. Zaleca się natomiast stosowanie uwierzytelniania MFA lub dłuższych haseł

- należy upewnić się czy użyte hasła nie należą do grupy powszechnie używanych, które zdążyły już trafić na listy popularnych haseł, co ułatwia ich złamanie. W tym celu można zablokować 1000 lub 10000 najczęściej używanych haseł, które spełniają wymagania dotyczące długości, ale znajdują się już na listach. Poniższy link zawiera szczegółową listę najpopularniejszych haseł:
https://github.com/danielmiessler/SecLists/tree/master/Passwords

Implementacja mechanizmu bezpiecznego odzyskiwania hasła

Większość aplikacji ma zwykle wbudowany odpowiedni mechanizm umożliwiający użytkownikowi uzyskanie dostępu do konta w przypadku zapomnienia hasła. Projektując funkcję odzyskiwania hasła warto wykorzystać elementy uwierzytelniania wieloskładnikowego. Można na przykład wprowadzić pytanie ochronne wymuszające podanie znanej już informacji, w oparciu o którą token wygeneruje kod lub hasło i prześle go na urządzenie.

Więcej szczegółowych informacji można znaleźć w: Forgot_Password_Cheat_Sheet (Ściągawka dot. rozwiązań dla opcji zapomniane hasło) oraz Choosing_and_Using_Security_Questions_Cheat_Sheet (Ściągawka nt. wyboru i użycia pytań ochronnych) .

Implementacja bezpiecznego przechowywania hasła

Aby zapewnić silne mechanizmy uwierzytelniania, aplikacja musi bezpiecznie przechowywać dane logowania każdego użytkownika. Ponadto powinno się wprowadzić kontrole kryptograficzne, tak aby w przypadku naruszenia danego poświadczenia (np. hasła) osoba atakująca nie uzyskała natychmiastowego dostępu do tych informacji.

Przykład jak przechowywać hasła w PHP

Poniżej znajduje się przykład bezpiecznego rozwiązania - hash w PHP - przy użyciu funkcji password_hash () dostępnej od wersji 5.5.0, która domyślnie używa algorytmu bcrypt. W przykładzie zastosowano współczynnik roboczy 15.

```
<?php
$cost = 15;
$password_hash = password_hash("secret_password", PASSWORD_DEFAULT, ["cost" =>
$cost] );
?>
```

Więcej informacji można znaleźć w OWASP Password Storage Cheat Sheet (Ściągawka OWASP dot. przechowywania danych).

Poziom 2: uwierzytelnianie wieloskładnikowe

Poziom AAL 2 dokumentu NIST 800-63b jest zarezerwowany dla aplikacji o podwyższonym ryzyku, które zawierają tzw. „self-assserted PII lub inne dane osobowe udostępnione online”. Na poziomie AAL 2 wymagane jest uwierzytelnianie wieloskładnikowe, w tym OTP (hasło jednorazowe) lub dowolne inne formy implementacji w oparciu o rozwiązania wieloskładnikowe.

Uwierzytelnianie wieloskładnikowe (MFA) zapewnia, że użytkownicy są faktycznie tymi, za których się podają, wymagając od nich identyfikacji za pomocą kombinacji, którą można pokrótce opisać jako:

- Coś, co znasz - hasło lub PIN
- Coś, co posiadasz - token lub telefon
- Coś, czym jesteś - dane biometryczne (np. odcisk palca)

Zabezpieczenia oparte wyłącznie na hasłach są dość słabe. Natomiast rozwiązania wieloskładnikowe są zdecydowanie bardziej niezawodne gdyż zmuszają osobę atakującą do zdobycia większej ilości informacji jeśli chce ona skutecznie przejść przez proces uwierzytelnienia w usłudze.

Warto zauważyć, że rozwiązania z wykorzystaniem biometrii nie są uznawane za akceptowalną formę cyfrowego uwierzytelniania jeśli wprowadza się je jako pojedynczy składnik zabezpieczeń. Jest to spowodowane tym, że dane biometryczne można pozyskać nawet bez wiedzy użytkownika np. w trybie online lub robiąc komuś zdjęcie telefonem wyposażonym w aparat (zdjęcie twarzy), zdejmując odciski palców z przedmiotów dotkniętych przez konkretną osobę (tzw. latent fingerprints) lub przechwycić je za pomocą obrazów o wysokiej rozdzielczości (np. wzór tęczówki oka). Dlatego też dane biometryczne mogą być wprowadzane jedynie w ramach uwierzytelniania wieloskładnikowego obok innego komponentu o charakterze fizycznym (vide: coś, co posiadasz). Można na przykład wprowadzić wieloskładnikowe rozwiązanie oparte na generowaniu jednorazowego hasła (OTP), które następnie użytkownik ręcznie wprowadzi do systemu w celu weryfikacji.

Poziom 3: uwierzytelnianie oparte na kryptografii

Zgodnie z NIST 800-63b, poziom 3 uwierzytelniania (AAL3) jest wymagany wtedy, gdy skutecznie przeprowadzony atak na system może prowadzić do powstania szkód osobistych, znacznych strat finansowych, szkodzić interesowi publicznemu lub wiązać się z naruszeniem prawa cywilnego bądź karnego. Poziom 3 (AAL3) wymaga uwierzytelnienia, które „polega na udowodnieniu, że posiada się klucz dostępu do systemu wykorzystującego protokół kryptograficzny". Ten rodzaj uwierzytelnienia wprowadza się po to, by zapewnić najwyższy poziom bezpieczeństwa danych. W tym celu zwykle stosuje się odpowiednie sprzętowe moduły kryptograficzne.

Zarządzanie sesją

Po pomyślnym uwierzytelnieniu użytkownika aplikacja może wybrać śledzenie i utrzymywanie aktualnego stanu uwierzytelnienia przez ograniczony czas. Umożliwi to użytkownikowi dalsze korzystanie z aplikacji bez konieczności ponownego uwierzytelniania przy wysyłaniu każdego żądania. Śledzenie stanu użytkownika to inaczej zarządzanie sesją.

Generowanie sesji i jej wygaśnięcie

Stan użytkownika jest śledzony w obrębie danej sesji. Każda sesja jest zwykle przechowywana na serwerze, po to by umożliwić również tradycyjne zarządzanie sesją w sieci. Identyfikator sesji jest następnie przekazywany użytkownikowi tak, aby mógł on zidentyfikować, która sesja po stronie serwera zawiera poprawne dane. Klient musi jedynie utrzymywać swój identyfikator sesji, który jednocześnie zachowuje poufne dane sesji po stronie serwera (lecz z dala od klienta).

Oto kilka możliwych kontroli bezpieczeństwa, które należy wziąć pod uwagę budując lub wdrażając rozwiązania w zakresie zarządzania sesją:

- Należy upewnić się, że identyfikator sesji jest długi, niepowtarzalny i losowy.

- Aplikacja powinna wygenerować nową sesję lub przynajmniej obrócić identyfikator sesji w trakcie uwierzytelniania lub ponownego uwierzytelniania.

- Aplikacja powinna wprowadzić limit czasu bezczynności oraz bezwzględną maksymalną długość trwania każdej sesji jako okresy, po których upłynięciu użytkownicy muszą ponownie się uwierzytelnić. Długość tych limitów powinna być odwrotnie proporcjonalna do wartości chronionych danych.

Więcej informacji na ten temat można znaleźć w: Session Management Cheat Sheet (Ściągawka nt. zarządzania sesją). Ponadto w rozdziale 3 dokumentu ASVS ujęto dodatkowe wymagania i zalecenia dotyczące zarządzania sesją.

Pliki cookie przeglądarki

Aplikacje internetowe wykorzystujące standardowe techniki zarządzania sesją zwykle przechowują indentyfikatory sesji w plikach cookie przeglądarki. Oto niektóre mechanizmy ochrony danych, które należy wziąć pod uwagę korzystając z plików cookie:

- Jeśli pliki cookie przeglądarki są używane jako mechanizm śledzenia sesji uwierzytelnionego użytkownika, powinny być one dostępne dla minimalnego zestawu domen i ścieżek, jak również należy oznaczyć je jako wygasające w okresie ważności sesji lub na krótko po nim.

- Należy ustawić flagę „bezpieczna/y” po to, by zapewnić transfer wyłącznie za pośrednictwem bezpiecznego kanału (TLS).

- Należy ustawić flagę HttpOnly, aby uniemożliwić dostęp do plików cookie przy pomocy JavaScript.

- Dodanie atrybutu „samesite” do plików cookie uniemożliwia niektórym nowoczesnym przeglądarkom wysyłanie plików cookie, zawierających żądania typu cross-site

(międzyserwisowe), zapewniając tym samym ochronę przed fałszywymi żądaniami lub atakami, które mogłyby prowadzić do wycieku informacji.

25

Tokeny

Sesje po stronie serwera mogą stanowić czynnik ograniczający niektóre formy uwierzytelniania. Natomiast tak zwane „usługi bezstanowe” pozwalają na zarządzanie danymi sesji przez klienta, zwiększając tym samym wydajność systemu gdyż serwer jest mniej obciążony przechowywanymi danymi oraz weryfikacją sesji użytkownika. Aplikacje „bezstanowe” generują krótkotrwały token dostępu, który może być użyty do uwierzytelnienia żądania klienta bez wysyłania poświadczeń użytkownika po pomyślnym początkowym uwierzytelnieniu.

JWT (Tokeny sieciowe JSON)

Token sieciowy JSON (JWT) to otwarty standard (RFC 7519), który definiuje zwarty i niezależny sposób bezpiecznego przesyłania informacji między stronami jako obiekt JSON. Informacje te można zweryfikować i traktować jako godne zaufania ponieważ są podpisywane cyfrowo. Token JWT jest tworzony podczas procesu uwierzytelniania i jest następnie weryfikowany przez serwer (lub serwery) przed rozpoczęciem przetwarzania informacji.

Jednak po ich utworzeniu tokeny JWT często nie są zapisywane przez serwer. Są one zwykle tworzone, a następnie przekazywane klientowi bez jakiegokolwiek zapisywania przez serwer. Integralność tokena jest utrzymywana za pomocą podpisów cyfrowych, dzięki czemu serwer może później zweryfikować, czy dany token JWT jest nadal ważny i czy nie został w jakikolwiek sposób zmodyfikowany od momentu jego utworzenia.

Takie rozwiązanie jest równocześnie bezstanowe i przenośne, więc nawet jeśli technologie klienta i serwera będą różne to nadal będą w stanie wejść w interakcję.

Uwaga

Cyfrowa tożsamość, uwierzytelnianie i zarządzanie sesją to zagadnienia bardzo rozległe. Zawarte w niniejszym dokumencie informacje obejmują wyłącznie kwestie podstawowe. Odpowiedzialność za bieżące utrzymywanie kompleksowych rozwiązań w zakresie uwierzytelniania i tożsamości cyfrowej należy powierzyć najbardziej utalentowanemu inżynierowi oprogramowania w swoim zespole.

Luki w zabezpieczeniach (prewencja)

OWASP Top 10 2017 A2- Broken Authentication and Session Management (informacje nt. nieskutecznych rozwiązań w zakresie uwierzytelniania oraz zarządzania sesją)
OWASP Mobile Top 10 2014-M5- Poor Authorization and Authentication (informacje nt. nieskutecznych rozwiązań w zakresie autoryzacji i uwierzytelniania opracowane dla aplikacji mobilnych)

26

Rekomendowane zasoby

OWASP Cheat Sheet: Authentication (Ściągawka nt. uwierzytelniania)
- OWASP Cheat Sheet: Password Storage (Ściągawka nt. metod przechowywania haseł)
- OWASP Cheat Sheet: Forgot Password (Ściągawka nt. projektowania rozwiązań dla funkcji: zapomniane hasło)
- OWASP Cheat Sheet: Choosing and Using Security Questions (Ściągawka nt. wyboru i używania pytań ochronnych)
- OWASP Cheat Sheet: Session Management (Ściągawka nt. zarządzania sesją)
- OWASP Cheat Sheet: IOS Developer (Ściągawka dla deweloperów IOS)
- OWASP Testing Guide: Testing for Authentication (Poradnik OWASP nt. testowania wprowadzanych rozwiązań uwierzytelniania)
- NIST Special Publication 800-63 Revision 3 - Digital Identity Guidelines (Wytyczne nt. tożsamości cyfrowej)

Narzędzia

Daniel Miessler: Most commonly found passwords (najczęściej używane hasła)

27

C7: Wymuszaj kontrolę dostępu

Charakterystyka

Kontrola dostępu (lub autoryzacja) to proces akceptowania lub odrzucania określonych żądań ze strony użytkownika, programu lub procesu. Kontrola dostępu obejmuje także przyznawanie i cofanie uprawnień.

Należy zauważyć, że autoryzacja (weryfikacja dostępu do określonych funkcji lub zasobów) nie jest równoważna z uwierzytelnianiem (weryfikacja tożsamości).

Funkcja *Kontrola Dostępu* często obejmuje wiele obszarów oprogramowania w zależności od stopnia złożoności całego systemu kontroli dostępu. Na przykład zarządzanie metadanymi

kontroli dostępu lub wbudowanie pamięci podręcznej, aby uzyskać efekt skalowalności to często stosowane dodatkowe komponenty w systemie kontroli dostępu.

Istnieje kilka różnych sposobów projektowania kontroli dostępu, które warto wziąć pod uwagę:

- Uznaniowa kontrola dostępu czyli DET (Discretionary Access Control) pozwala ograniczyć dostęp do obiektów (np. plików, jednostek danych) w oparciu o tożsamość oraz tzw. need-to-know podmiotów (np. użytkowników, procesów) i/lub grup, do których należy obiekt.

- Obowiązkowa kontrola dostępu czyli MAC (Mandatory Access Control) to sposób ograniczania dostępu do zasobów systemowych w oparciu o wrażliwość (reprezentowaną przez etykietę) informacji zawartych w zasobach systemowych oraz formalną autoryzację (tj. certyfikat bezpieczeństwa) użytkowników, chcących uzyskać dostęp do informacji o takiej wrażliwości.

- Kontrola dostępu oparta na rolach czyli RBAC (Role Based Access Control) to model kontroli dostępu do zasobów, w którym dozwolone działania na zasobach są identyfikowane w powiązaniu z rolami, a nie z indywidualną tożsamością podmiotów.

- Kontrola dostępu oparta na atrybutach czyli ABAC (Attribute Based Access Control) akceptuje lub odrzuca żądania użytkowników w oparciu o dowolne atrybuty użytkownika i dowolne atrybuty obiektu, a także warunki środowiska, które mogą zostać uznane w skali globalnej i które są zdecydowanie bardziej adekwatne w kontekście obowiązujących na świecie polityk bezpieczeństwa.

Zasady projektowania kontroli dostępu

Następujące „dobre” zalecenia dotyczące projektowania kontroli dostępu należy wziąć pod uwagę już na początkowych etapach tworzenia aplikacji.

1.) Zaprojektuj kontrolę dostępu na początku tworzenia aplikacji

Po wybraniu określonego wzorca projektowania systemu kontroli dostępu wprowadzenie jakichkolwiek zasadniczych zmian jest często trudnym i czasochłonnym przedsięwzięciem. Kontrola dostępu jest jednym z głównych elementów tworzenia zabezpieczeń aplikacji, dlatego też powinna zostać zaprojektowana z góry, zwłaszcza jeśli ma spełniać szczególne wymagania w zakresie kontroli dostępu np. obsługiwać wielu użytkowników (multi-tenancy) czy uwzględniać kontrolę horyzontalną (zależną od danych).

Projekt systemu kontroli dostępu może początkowo wydawać się prosty i nieskomplikowany, ale często, już w trakcie tworzenia, przekształca się w złożone rozwiązanie z zaawansowaną kontrolą zabezpieczeń. Oceniając możliwości kontroli dostępu w ramach oprogramowania,

należy upewnić się, że funkcja kontroli dostępu będzie pozwalać na dostosowanie jej do specyficznych potrzeb.

2.) Wymuszaj, aby wszystkie żądania przechodziły przez kontrolę dostępu

Warto upewnić się czy aby na pewno wszystkie żądania przechodzą przez dowolną warstwę weryfikacji kontroli dostępu. Technologie takie jak filtry Java lub inne automatyczne mechanizmy przetwarzania żądań są idealnymi artefaktami programistycznymi, dzięki którym można mieć pewność, że żądania przechodzą przez dowolny rodzaj kontroli dostępu.

3.) Domyślnie odmawiaj dostępu

Domyślna odmowa jest zasadą w oparciu, o którą przyjmuje się, że jeśli żądanie nie należy do konkretnej grupy dozwolonych, nastąpi odmowa dostępu. Reguła ta może przejawiać się w kodzie aplikacji w przeróżny sposób. Oto kilka przykładów:

1. Kod aplikacji może zgłaszać błąd lub wyjątek podczas przetwarzania żądań kontroli dostępu. W takich przypadkach zawsze powinna nastąpić odmowa dostępu.
2. Kiedy administrator tworzy nowego użytkownika lub jeśli użytkownik zarejestruje się na nowym koncie, takie konto powinno mieć domyślnie jedynie minimalny dostęp lub jego całkowity brak aż do momentu skonfigurowania tego dostępu.
3. Po dodaniu nowej funkcji do aplikacji wszyscy użytkownicy powinni zostać pozbawieni możliwości korzystania z tej funkcji do czasu jej prawidłowej konfiguracji.

4.) Zasada najmniejszego przywileju

Upewnij się, że wszyscy użytkownicy, programy lub procesy mają wyłącznie najmniejszy możliwy lub niezbędny dostęp. Należy uważać na systemy, które nie zapewniają możliwości konfiguracji granularnej kontroli dostępu.

5.) Nie stosuj trwałego kodowania ról

Wiele struktur (frameworków) aplikacji posiada domyślną kontrolę dostępu opartą na rolach. Powszechnie spotyka się kody aplikacji wypełnione zapisami jak poniżej:

```
if (user.hasRole("ADMIN")) || (user.hasRole("MANAGER")) { deleteAccount();
}
```

Należy uważać na taki rodzaj programowania opartego na rolach. Ma on bowiem następujące ograniczenia i potencjalne zagrożenia:

- Taki rodzaj programowania opartego na rolach jest przede wszystkim kruchy. Łatwo jest utworzyć nieprawidłowe lub brakujące sprawdzanie ról w kodzie.

- Programowanie oparte na rolach nie pozwala na obsługę wielu użytkowników (multi-tenancy). Aby umożliwić systemom opartym na rolach stosowanie różnych reguł dla różnych klientów będzie trzeba wprowadzić środki zasadniczo ekstremalne, jak na przykład rozwidlenie kodu lub dodatkowe sprawdzenia każdego klienta.

- Programowanie oparte na rolach nie umożliwia stosowania reguł kontroli dostępu opartych na danych czy reguł horyzontalnych.

- Zarówno audyt jak i weryfikacja ogólnej polityki kontroli dostępu do aplikacji mogą być utrudnione w przypadku dużych baz kodów obejmujących wiele poziomów sprawdzania kontroli dostępu.

Zamiast tego warto rozważyć zastosowanie następującej metodologii programowania kontroli dostępu:

```
if (user.hasAccess ("DELETE_ACCOUNT")) {deleteAccount ();
}
```

Takie sprawdzanie kontroli dostępu oparte na atrybutach lub funkcjach stanowi punkt wyjścia do budowy dobrze zaprojektowanych systemów kontroli dostępu z bogatym wachlarzem funkcji. Tego typu programowanie pozwala również z czasem zwiększyć zdolność dostosowywania kontroli dostępu.

6.) Rejestruj wszystkie zdarzenia kontroli dostępu

Wszelkie awarie kontroli dostępu powinny być rejestrowane, ponieważ mogą one wskazywać na aktywność złośliwego użytkownika, który sonduje aplikację pod kątem luk.

Luki w zabezpieczeniach (prewencja)

OWASP Top 10 2017-A5-Broken Access Control (informacje nt. nieskutecznych rozwiązań w zakresie kontroli dostępu)
OWASP Mobile Top 10 2014-M5 Poor Authorization and Authentication (informacje nt. nieskutecznych rozwiązań w zakresie autoryzacji i uwierzytelniania opracowane dla aplikacji mobilnych)

Rekomendowane zasoby

- OWASP Cheat Sheet: Access Control (Ściągawka nt. kontroli dostępu)
- OWASP Cheat Sheet: iOS Developer - Poor Authorization and Authentication (Ściągawka dla deweloperów iOS nt. nieskutecznych rozwiązań w zakresie autoryzacji i uwierzytelniania)
- OWASP Testing Guide: Testing for Authorization (Poradnik OWASP nt. testowania wprowadzanych rozwiązań dot. autoryzacji)

Narzędzia

OWASP ZAP with the optional Access Control Testing add-on (OWASP ZAP z opcjonalnym dodatkiem nt. testowania kontroli dostępu)

C8: Chroń dane wszędzie

Charakterystyka

Dane wrażliwe, takie jak hasła, numery kart kredytowych, dane dotyczące zdrowia, dane osobowe i tajemnice handlowe wymagają dodatkowej ochrony, szczególnie jeśli dane te podlegają przepisom prawa dotyczącym ochrony prywatności (ogólne rozporządzenie UE w sprawie ochrony danych - GDPR), przepisom dotyczącym ochrony danych finansowych, jak na przykład PCI Data Security Standard (PCI DSS) lub innym regulacjom prawnym.

Osoby atakujące mogą wykraść dane z aplikacji internetowych i usług internetowych na wiele sposobów. Na przykład jeśli poufne informacje są przesyłane przez Internet bez zabezpieczeń komunikacyjnych, osoba atakująca w ramach udostępnionego połączenia bezprzewodowego może zobaczyć i wykraść dane innego użytkownika. Ponadto osoba atakująca może użyć funkcji SQL Injection do kradzieży haseł i innych danych uwierzytelniających z bazy danych aplikacji, a następnie ujawnić te informacje publicznie.

Klasyfikacja danych

Bardzo ważne jest klasyfikowanie danych w systemie i określanie ich poziomu wrażliwości. Każdą kategorię danych można następnie odwzorować na reguły ochrony niezbędne dla każdego poziomu poufności. Na przykład publiczne informacje marketingowe, które nie są wrażliwe, można sklasyfikować jako dane publiczne, które można umieścić na publicznej stronie internetowej. Numery kart kredytowych mogą być klasyfikowane jako prywatne dane użytkowników, które mogą wymagać zaszyfrowania podczas ich przechowywania lub przesyłania.

Szyfrowanie danych w tranzycie

Podczas transmisji poufnych danych przez dowolną sieć, należy rozważyć wprowadzenie całościowego zabezpieczenia komunikacji (lub szyfrowanie w tranzycie). TLS jest zdecydowanie najczęstszym i bardzo szeroko obsługiwanym protokołem kryptograficznym zapewniającym bezpieczeństwo komunikacji. Jest aktualnie używany przez wiele typów aplikacji (aplikacje i usługi internetowe, aplikacje mobilne) do bezpiecznego komunikowania się poprzez sieć. Protokół TLS musi być odpowiednio skonfigurowany, aby móc zapewnić bezpieczną komunikację.

Podstawową korzyścią protokołu TLS (Transport Layer Security) jest ochrona danych aplikacji internetowych przed nieautoryzowanym ujawnianiem i modyfikowaniem podczas przesyłania danych między klientami (przeglądarki), a serwerem aplikacji internetowej oraz między serwerem aplikacji WWW, a innymi komponentami niepochodzącymi z przeglądarki.

Szyfrowanie danych w spoczynku

Pierwszą zasadą zarządzania danymi wrażliwymi jest, o ile to możliwe, unikanie ich przechowywania. Jeśli jednak dane wrażliwe muszą być przechowywane, wtedy należy upewnić się, czy są chronione szyfrem, po to by uniknąć nieautoryzowanego ujawnienia i modyfikacji danych.

Techniki szyfrowania należą do najbardziej zaawansowanych kwestii w kontekście ochrony informacji, a ich zrozumienie wymaga dużej wiedzy i doświadczenia. Istnieje wiele możliwych podejść i technik szyfrowania, z których każda ma swoje wady i zalety. Architekci oraz programiści rozwiązań internetowych muszą swobodnie poruszać się w tym obszarze i dokładnie rozumieć zasady stosowanych technik kryptograficznych. Ponadto wszelkie istotne badania naukowe na temat kryptografii opierają się zazwyczaj na zaawansowanej matematyce i teorii liczb, co stanowi poważną barierę wejścia.

Zamiast budować funkcje kryptograficzne od podstaw, zdecydowanie zaleca się korzystanie z recenzowanych i otwartych rozwiązań, takich jak projekt Google Tink, Libsodium i bezpieczna pamięć masowa wbudowana w wiele frameworków programowych i usług w chmurze.

Aplikacja mobilna: bezpieczne przechowywanie lokalne

Aplikacje mobilne są szczególnie narażone na wyciek danych, ponieważ często dochodzi do zgubienia lub kradzieży urządzeń mobilnych, przy czym zwykle zawierają one dane wrażliwe.

Zasadniczo na urządzeniu mobilnym powinno się przechowywać tylko niezbędne minimum danych. Ale jeśli konieczne jest przechowywanie danych wrażliwych na urządzeniu

mobilnym, powinny być one przechowywane w odpowiednim katalogu, charakterystycznym dla konkretnego systemu operacyjnego. W systemie Android będzie to pęk kluczy systemu Android, a w systemie iOS będzie to pęk kluczy iOS.

Cykl życia kluczy

Klucze poufne są używane w aplikacjach o dużej liczbie ważnych funkcji. Na przykład mogą być używane do podpisywania znacznika JWT, ochrony kart kredytowych czy zapewniania różnych form uwierzytelniania. Ułatwiają również wprowadzenie różnego rodzaju poufnych zabezpieczeń. Zarządzając kluczami należy jednak przestrzegać kilku zasad, w tym:

- Upewnić się, że każdy poufny klucz jest chroniony przed nieautoryzowanym dostępem
- Przechowywać klucze w odpowiednim skarbcu poufnych danych, jak opisano poniżej
- Używać niezależnych klawiszy, gdy wymagane jest użycie wielu klawiszy
- Utworzyć odpowiednie struktury wspierające, które w razie potrzeby będą zmieniać algorytmy i klucze
- Utworzyć takie funkcje aplikacji, które będą obsługiwać rotację klawiszy

Zarządzanie poufnością w aplikacji

Aplikacje zawierają liczne „dane ukryte”, które są potrzebne do przeprowadzenia operacji chronionych. Należą do nich certyfikaty, hasła połączenia SQL, poświadczenia konta usługi zewnętrznej, hasła, klucze SSH, klucze szyfrowania i inne. Nieautoryzowane ujawnienie lub modyfikacja tych danych może doprowadzić do naruszenia całego systemu bezpieczeństwa. Należy zatem rozważyć następujące kwestie:

- Nie przechowywać ich w kodzie, plikach konfiguracyjnych ani nie przekazywać ich za pomocą zmiennych środowiskowych. Warto korzystać z narzędzi takich jak GitRob lub TruffleHog do skanowania repozytoriów.

- Przechowywać klucze i inne dane poziomu aplikacji w skarbcu poufnych kluczy, np. w KeyWhiz, Vault Project firmy Hashicorp lub Amazon KMS, aby zapewnić bezpieczne przechowywanie i dostęp do tych danych na poziomie aplikacji w trakcie działania programu.

Luki w zabezpieczeniach (prewencja)

OWASP Top 10 2017 - A3: Sensitive Data Exposure (informacje nt. ekspozycji danych wrażliwych)
OWASP Mobile Top 10 2014-M2 Insecure Data Storage (informacje nt. niezabezpiecznonego przechowywania danych w aplikacjach mobilnych)

Rekomendowane zasoby

- OWASP Cheat Sheet: Transport Layer Protection (Ściągawka nt. protokołu TLS)
- Ivan Ristic: SSL/TLS Deployment Best Practices (Zestaw najlepszych praktyk związanych z protokołem SSL/TLS)
- OWASP Cheat Sheet: HSTS (Ściągawka nt. HSTS - Http Strict Transport Security)
- OWASP Cheat Sheet: Cryptographic Storage (Ściągawka nt. szyfrowanego przechowywania danych)
- OWASP Cheat Sheet: Password Storage (Ściągawka nt. przechowywania haseł)
- OWASP Cheat Sheet: IOS Developer - Insecure Data Storage (informacje nt. niezabezpiecznonego przechowywania danych skierowane do deweloperów iOS)
- OWASP Testing Guide: Testing for TLS (Poradnik OWASP nt. testowania wprowadzanych rozwiązań TLS)

Narzędzia

- SSLyze - Biblioteka skanowania konfiguracji SSL i narzędzie CLI
- SSLLabs - Bezpłatna usługa skanowania i sprawdzania konfiguracji TLS/SSL
- OWASP O-Saft TLS Tool - Narzędzie do testowania połączenia TLS
- GitRob - Narzędzie wiersza poleceń do wyszukiwania poufnych informacji w publicznie dostępnych plikach na GitHub
- TruffleHog - Przegląd danych ukrytych przypadkowo
- KeyWhiz - tzw. Secrets manager
- Hashicorp Vault - tzw. Secrets manager
- Amazon KMS - Zarządzanie kluczami w Amazon AWS

C9: Wprowadź rejestrowanie i monitorowanie bezpieczeństwa systemu

Charakterystyka

Rejestrowanie zdarzeń to koncepcja, z której większość programistów korzysta już od dawna w celu debugowania i diagnostyki. Rejestrowanie zdarzeń dotyczących zabezpieczeń jest równie podstawowym rozwiązaniem, mającym na celu rejestrowanie informacji o zabezpieczeniach w trakcie działania aplikacji. Monitorowanie to przegląd na żywo dzienników aplikacji i zabezpieczeń przy użyciu różnych form automatyzacji. Te same narzędzia i wzorce mogą być używane do operacji, debugowania i w celach bezpieczeństwa.

Korzyści z rejestrowania zabezpieczeń

Rejestrowanie bezpieczeństwa może być wykorzystywane do:

1) Wspierania systemów wykrywania włamań
2) Analizy w kontekście informatyki śledczej
3) Spełnienia wymogów zgodności z przepisami

Implementacja rejestrowania zabezpieczeń

Poniżej znajduje się lista najlepszych praktyk wdrażania protokołów bezpieczeństwa.

- Należy stosować wspólny format rejestrowania w systemie oraz w systemach organizacji. Przykładem typowej struktury logowania jest usługa rejestrowania Apache (Apache Logging Services), która pozwala zapewnić spójność rejestrowania zdarzeń między aplikacjami Java, PHP, .NET i C ++.

- Nie powinno się rejestrować ani zbyt wielu, ani zbyt niewielu zdarzeń. Warto na przykład pamiętać, aby zawsze rejestrować znacznik czasu i informacje identyfikujące, w tym źródłowy adres IP i identyfikator użytkownika, ale należy uważać, aby nie rejestrować prywatnych lub poufnych danych.

- Warto zwrócić szczególną uwagę na synchronizację czasu między węzłami, aby zapewnić zgodność sygnatur czasowych.

Rejestrowanie w celu wykrycia włamań i odpowiedzi systemu

Należy stosować rejestrowanie zdarzeń, tak aby móc zidentyfikować ewentualne złośliwe działania użytkownika. Warto przede wszystkim rejestrować:

- Przesłane dane spoza oczekiwanego zakresu liczbowego.

- Przesłane dane, które dotyczą zmian takich danych, których nie można modyfikować (np. lista wyboru, pole wyboru lub inny ograniczony element wejścia).

- Żądania naruszające reguły kontroli dostępu po stronie serwera.

- Bardziej obszerna lista możliwych punktów detekcji jest dostępna tutaj.

Gdy aplikacja napotka takie zdarzenie, powinna je przynajmniej zarejestrować oraz oznaczyć jako poważny problem. W idealnym przypadku aplikacja powinna także reagować na możliwy zidentyfikowany atak, na przykład unieważniając sesję użytkownika i blokując konto użytkownika. Mechanizmy odpowiedzi pozwalają oprogramowaniu reagować w czasie rzeczywistym na ewentualne zidentyfikowane ataki.

Projektowanie rejestru zabezpieczeń

Rozwiązania rejestrujące muszą być tworzone i zarządzane w sposób bezpieczny. Projekt bezpiecznego logowania może obejmować:

- Kodowanie i zatwierdzanie wszelkich niebezpiecznych znaków przed rejestracją zdarzenia, aby zapobiec atakom na dziennik zdarzeń typu Log Injection lub Log Forging.

- Odmowę rejestracji informacji wrażliwych. Na przykład nie powinno się rejestrować hasła, identyfikatora sesji, kart kredytowych ani numerów ubezpieczenia społecznego.

- Ochronę spójności dziennika. Osoba atakująca może próbować manipulować i fałszować dane w dziennikach. W związku z tym należy uwzględnić możliwość zezwolenia na audyt plików i zmian dziennika.

- Przesyłanie dzienników zdarzeń z rozproszonych systemów plików do centralnej, bezpiecznej usługi rejestrowania. Dzięki temu dane dziennika nie zostaną utracone po zaatakowaniu jednego węzła. Takie rozwiązanie pozwala również wprowadzić scentralizowane monitorowanie.

Rekomendowane zasoby

- OWASP AppSensor Detection Points - Punkty wykrywania używane do identyfikacji złośliwego użytkownika sondującego luki lub słabe punkty aplikacji.
- OWASP Log injection
- OWASP Log forging
- OWASP Cheat Sheet: Logging - Jak prawidłowo zaimplementować rejestr zdarzeń w aplikacji
- OWASP Development Guide: Logging
- OWASP Code Review Guide: Reviewing Code for Logging Issues (Poradnik dot. sprawdzania kodu pod kątem problemów z rejestrem zdarzeń)

Narzędzia

- OWASP Security Logging Project
- Apache Logging Services

C10: Usuń wszelkie błędy i wyjątki

Charakterystyka

Obsługa wyjątków jest koncepcją programistyczną, która pozwala aplikacji odpowiednio reagować na różnego rodzaju błędy (np. brak połączenia z siecią lub z bazą danych itp.). Właściwe obchodzenie się z wyjątkami i błędami ma kluczowe znaczenie dla zapewnienia niezawodności i bezpieczeństwa kodu.

Obsługa błędów i wyjątków występuje we wszystkich obszarach aplikacji, w tym w „krytycznej" warstwie logiki biznesowej, a także w funkcjach zabezpieczeń i kodzie źródłowym.

Obsługa błędów jest również ważna z perspektywy wykrywania włamań do systemu. Niektóre ataki na aplikację mogą powodować zaistnienie błędów, które mogą okazać się przydatne w wykrywaniu ataków w toku.

Usterki w obsłudze błędów

Naukowcy z Uniwersytetu w Toronto odkryli, że nawet małe usterki w obsłudze błędów lub zapomnienie o konieczności obsługi błędów mogą prowadzić do katastrofalnych awarii w systemach rozproszonych.

Usterki w obsłudze błędów mogą również prowadzić do różnego rodzaju luk w zabezpieczeniach:

- **Wyciek informacji**: Wyciekanie poufnych informacji w komunikatach o błędach może w niezamierzony sposób pomóc osobie atakującej. Na przykład błąd, który zwraca tzw. stack trace (dosłownie: ślad stosu) lub inne wewnętrzne szczegóły błędu, może dostarczyć osobie atakującej informacji o środowisku systemowym. Nawet niewielkie różnice w obsłudze warunków błędu (np. zwracanie komunikatu „nieprawidłowy użytkownik" lub „nieprawidłowe hasło" w przypadku błędów uwierzytelniania) mogą dostarczyć cennych wskazówek osobom atakującym. Jak opisano powyżej, należy przede wszystkim pamiętać o stosowaniu rejestru szczegółów błędów dla ewentualnych celów śledczych i debugowania, ale nie wolno ujawniać tych informacji, szczególnie zewnętrznemu klientowi.

- **Obejście TLS**: Błąd goto „fail bug" to błąd przepływu sterowania (control flow error) w kodzie obsługi błędów, który doprowadził do całkowitego uszkodzenia połączeń TLS w systemach Apple.

- **DOS**: Brak podstawowej obsługi błędów może doprowadzić do zamknięcia systemu. Luka takiego rodzaju może w dość łatwy sposób zostać wykorzystana przez osoby atakujące. Inne problemy z obsługą błędów mogą np. prowadzić do zwiększonego wykorzystania procesora (CPU) lub dysku w sposób, który może pogorszyć działanie całego systemu.

Kilka dobrych porad

- Należy zarządzać wyjątkami w sposób scentralizowany, aby uniknąć powielania bloków typu try/catch w kodzie. Warto też sprawdzić czy wszystkie nieoczekiwane zdarzenia są poprawnie obsługiwane wewnątrz aplikacji.

- Należy upewnić się, że komunikaty o błędach wyświetlane użytkownikom nie powodują wycieku ważnych danych, a jednocześnie są wystarczająco szczegółowe, by umożliwić prawidłową reakcję użytkownika.

- Warto sprawdzić czy wyjątki są rejestrowane w sposób zapewniający wystarczającą ilość informacji umożliwiając tym samym uzyskanie odpowiedniego wsparcia, lecz również kontrolę jakości, analizę śledczą lub powołanie zespołów reagowania w celu zrozumienia problemu.

- Należy dokładnie przetestować i zweryfikować kod obsługi błędów.

Rekomendowane zasoby

- OWASP Code Review Guide: Error Handling (Poradnik zawierający informacje nt. obsługi błędów)
- OWASP Testing Guide: Testing for Error Handling (Poradnik zawierający informacje nt. testowania rozwiązań do obsługi błędów)
- OWASP Improper Error Handling (Zawiera informacje nt. niewłaściwej obsługi błędów)
- CWE 209: Information Exposure Through an Error Message (Informacje nt. ujawniania informacji poprzez komunikat o błędzie)
- CWE 391: Unchecked Error Condition (Informacje nt. niesprawdzonych warunków zaistnienia błędów)

Narzędzia

Error Prone - narzędzie analizy statycznej od Google, służące do wykrywania typowych usterek w obsłudze błędów dla programistów Java

Jednym z najbardziej znanych automatycznych narzędzi do wykrywania błędów w środowisku wykonawczym jest Chaos Monkey Netflixa, który celowo wyłącza konkretne instancje systemu w taki sposób, żeby cała usługa została odzyskana poprawnie.

Kilka słów na koniec

Ten dokument powinien być postrzegany jako punkt wyjścia, a nie kompleksowy zestaw dostępnych technik i praktyk. Pragniemy podkreślić raz jeszcze, że niniejszy dokument ma na celu dostarczenie podstawowej wiedzy na temat budowy bezpiecznego oprogramowania.

A oto kolejne kroki, które warto podjąć, by pomyślnie zbudować program zabezpieczeń aplikacji:

1. Aby zrozumieć niektóre z zagrożeń związanych z bezpieczeństwem aplikacji internetowych, warto szczegółowo zapoznać się z Owasp Top 10 oraz Owasp Mobile Top 10.
2. Zgodnie z Proactive Control # 1, program zabezpieczeń powinien zawierać kompleksową listę wymagań bezpieczeństwa opartych na standardach, takich jak np. OWASP (Web) ASVS czy OWASP (Mobile) MASVS.
3. Aby zrozumieć podstawowe elementy składowe bezpiecznego oprogramowania z punktu widzenia makro, warto zapoznać się z projektem OWASP OpenSAMM.

W przypadku jakichkolwiek pytań do zespołu kierującego projektem, proszę zarejestrować się na naszej liście dyskusyjnej dostępnej pod adresem: https://lists.owasp.org/mailman/listinfo/owasp_proactive_controls.