PAWEŁ RUS

PIOTR KONDZIOŁKA

SZCZEPAN DWORAK

CTF: Praktyczna nauka bezpieczeństwa aplikacji webowych

Dlaczego bezpieczeństwo aplikacji webowych jest ważne?

Aplikacje webowe obsługują wrażliwe dane, takie jak dane osobowe, loginy czy informacje finansowe, co czyni je częstym celem cyber ataków . Zaniedbania w ich zabezpieczeniu mogą prowadzić do wycieków danych, strat finansowych czy utraty zaufania użytkowników. Umiejętności związane z ochroną aplikacji można rozwijać praktycznie, np. biorąc udział w konkursach CTF (Capture The Flag) dostępnych na platformach takich jak TryHackMe, które oferują scenariusze odzwierciedlające rzeczywiste zagrożenia.

Rola CTF w edukacji bezpieczeństwa

Praktyczna nauka: Uczestnicy zdobywają doświadczenie w realnych scenariuszach zagrożeń.

Zrozumienie podatności: Analizowanie i wykorzystywanie luk w kontrolowanym środowisku pomaga lepiej rozumieć mechanizmy ataków.

Rozwój umiejętności technicznych: Poprawia umiejętności z zakresu kryptografii, inżynierii odwrotnej, eksploitacji i analizy kodu.

Praca zespołowa: CTF-y często wymagają współpracy i komunikacji w grupie.

Motywacja do nauki: Element rywalizacji sprawia, że uczestnicy chętniej poszerzają wiedzę.

Budowanie świadomości bezpieczeństwa: Pomaga lepiej zabezpieczać aplikacje i systemy.

Kluczowe aspekty bezpieczeństwa

Uwierzytelnianie i autoryzacja: Bezpieczne hasła, kontrola dostępu.

Walidacja danych wejściowych: Ochrona przed XSS i SQL Injection.

Bezpieczne sesje: Ciasteczka (Secure, HttpOnly), unikanie sesji w URL.

Szyfrowanie: HTTPS (SSL/TLS), szyfrowanie danych w tranzycie i spoczynku.

Zarządzanie lukami: Testy penetracyjne, szybkie aktualizacje.

Bezpieczny kod: Przeglądy i unikanie niebezpiecznych funkcji.

Monitorowanie: Logi i alerty podejrzanych działań.

SQL Injection

To atak polegający na wstrzyknięciu złośliwego kodu SQL do zapytań wysyłanych do bazy danych. Najczęściej wykorzystuje podatności w walidacji danych wejściowych. Skutkiem takiego ataku może być kradzież danych, manipulacja nimi lub przejęcie kontroli nad systemem. Aby zapobiegać atakom, należy stosować parametryzowane zapytania, unikać konkatenacji danych użytkownika w SQL oraz regularnie testować aplikację pod kątem podatności.

Przykład ataku SQL Injection

```
$query = "SELECT * FROM users WHERE username = '$username' AND password = '$password'";
```

Atak: Użytkownik wprowadza w polu "hasło" następujący ciąg: ' OR '1'='1

Zmieniony zapytanie SQL: SELECT * FROM users WHERE username = 'admin' AND password = '' OR '1'='1';

Skutek:

Zapytanie zwraca dane użytkownika "admin" niezależnie od hasła, ponieważ warunek '1'='1' zawsze jest prawdziwy.

Cross-Site Scripting (XSS)

To atak polegający na wstrzyknięciu złośliwego kodu (np. JavaScript) do treści wyświetlanej przez aplikację webową. Atakujący wykorzystuje tę lukę, aby kraść dane użytkowników, przechwytywać sesje lub wykonywać działania w ich imieniu. Aby zapobiegać XSS, należy prawidłowo filtrować i kodować dane wejściowe oraz stosować polityki zabezpieczeń, takie jak Content Security Policy (CSP).

Przykład ataku XSS

W przypadku podatności na XSS, atakujący może wstrzyknąć złośliwy kod JavaScript, który zostanie wykonany przez przeglądarkę użytkownika.

Podatny kod (brak walidacji danych wejściowych):

<input type="text" name="comment" value="<?php echo $_GET['comment']; ?>" />

Atak: Użytkownik wprowadza w polu komentarza następujący ciąg:

<script>alert('Zostałeś zaatakowany!');</script>

Skutek:

Przeglądarka użytkownika wykona wstrzyknięty kod, wyświetlając alert z komunikatem "Zostałeś zaatakowany!". W bardziej zaawansowanych atakach, atakujący może kraść ciasteczka sesyjne lub przejąć konto użytkownika.

Cross-Site Request Forgery (CSRF)

CSRF to atak, w którym złośliwa strona internetowa wysyła nieautoryzowane żądania do aplikacji, w której użytkownik jest zalogowany. Atakujący wykorzystuje to, by wykonać niepożądane akcje w imieniu ofiary, takie jak zmiana ustawień konta lub wykonanie transakcji. Aby chronić się przed CSRF, należy używać tokenów CSRF w formularzach, weryfikować źródło żądań i stosować metody autoryzacji, które nie polegają na prostym ciasteczku sesyjnym.

Przykład ataku CSRF

Atak CSRF polega na wysyłaniu nieautoryzowanych żądań w imieniu zalogowanego użytkownika. Aplikacja, która nie weryfikuje źródła żądania, może być podatna na ten typ ataku.

Podatny kod (brak weryfikacji CSRF):

```
<form action="change_password.php" method="POST">

  <input type="text" name="new_password" value="newpassword123" />

  <input type="submit" value="Change Password" />

</form>
```

Atak: Atakujący może stworzyć złośliwą stronę z poniższym kodem HTML:

```
<img src="http://example.com/change_password.php?new_password=attackerpassword" />
```

Skutek:

Jeśli ofiara odwiedzi stronę atakującego, jej przeglądarka wyśle nieautoryzowane żądanie do aplikacji, zmieniając hasło na "attackerpassword", bez jej wiedzy.

Ochrona:

Użycie tokenów CSRF w formularzach i weryfikacja źródła żądania zapobiega takim atakom.

Python Code Injection

Python Code Injection to atak polegający na wstrzyknięciu i wykonaniu złośliwego kodu Python w aplikacji webowej. Dzieje się tak, gdy dane wejściowe użytkownika są niewłaściwie przetwarzane przez funkcje wykonujące kod dynamicznie, takie jak `eval()`, `exec()` czy inne metody manipulujące kodem.

Skutki:

- Wykonanie dowolnego kodu przez atakującego.
- Kradzież danych wrażliwych lub przejęcie aplikacji.
- Możliwość eskalacji uprawnień w systemie.

Przykład ataku Python Code Injection

Podatny kod:

```
def run_code(user_input):
    eval(user_input)
```

```
run_code("print('Zostałeś zaatakowany!')")
```

Atak:

```
__import__('os').system('rm -rf /')
```

Ochrona przed Python Code Injection

- Unikanie funkcji takich jak eval(), exec() i compile() przy przetwarzaniu danych wejściowych.
- Stosowanie bibliotek do parsowania danych wejściowych (np. ast.literal_eval do bezpiecznego parsowania Python literals).
- Walidacja i sanitizacja danych wejściowych.
- Używanie mechanizmów separacji kodu, takich jak sandboxing.

OS Command Injection

OS Command Injection to atak polegający na wstrzyknięciu złośliwego polecenia systemowego do aplikacji, która wykonuje polecenia w systemie operacyjnym. Może wystąpić, gdy dane wejściowe użytkownika są bezpośrednio włączane w wywołania systemowe.

Skutki

- Kradzież danych lub przejęcie systemu.
- Zmiana plików i konfiguracji serwera.
- Uruchamianie programów lub skryptów przez atakującego.

Przykład ataku OS Command Injection

Podatny kod:

```
import os
os.system(f"ping {user_input}")
```

Atak:

Użytkownik wprowadza:

```
127.0.0.1 && rm -rf /
```

Ochrona przed OS Command Injection

- Używanie dedykowanych bibliotek do obsługi systemowych funkcji (np. subprocess.run z opcją shell=False).
- Walidacja i sanitizacja danych wejściowych.
- Unikanie włączania danych użytkownika bezpośrednio w polecenia systemowe.
- Izolacja środowiska aplikacji, np. konteneryzacja.

Server-Side Template Injection (SSTI)

Definicja

Server-Side Template Injection (SSTI) to atak polegający na wstrzyknięciu złośliwego kodu do szablonów renderowanych po stronie serwera. Dzieje się tak, gdy dane wejściowe użytkownika są niewłaściwie przetwarzane przez silnik szablonów, taki jak Jinja2, Twig czy Velocity.

Skutki

Wykonanie dowolnego kodu na serwerze.

Ujawnienie wrażliwych danych, takich jak klucze API, dane konfiguracyjne czy bazy danych.

Możliwość eskalacji uprawnień i przejęcia pełnej kontroli nad systemem.

Przykład ataku SSTI

```
from flask import Flask, request, render_template_string


app = Flask(__name__)


@app.route('/greet', methods=['GET'])
def greet():
    user = request.args.get('user', 'World')
    return render_template_string(f"Hello {user}!")
```

Ochrona przed SSTI

Używanie dedykowanych funkcji do renderowania szablonów, które nie dopuszczają do dynamicznego wstrzykiwania kodu.

Walidacja i sanitizacja danych wejściowych od użytkowników.

Unikanie włączania niesprawdzonych danych użytkownika w kod szablonów.

Regularne testy aplikacji pod kątem podatności SSTI i aktualizowanie silników szablonów.

Podsumowanie

Kluczowe wnioski:

Bezpieczeństwo aplikacji webowych to fundamentalny aspekt współczesnych systemów online.

Zrozumienie podatności, takich jak SQL Injection, XSS, CSRF, Python Code Injection czy OS Command Injection, pozwala nie tylko lepiej je wykrywać, ale także skuteczniej im zapobiegać.

CTF-y to świetna forma nauki poprzez praktykę, która buduje świadomość i rozwija umiejętności potrzebne do ochrony aplikacji.

Dlaczego to ważne?

W miarę rozwoju technologii rośnie także kreatywność atakujących.

Każdy, kto tworzy, rozwija lub zabezpiecza aplikacje, ma realny wpływ na bezpieczeństwo użytkowników i systemów.

Nasze przesłanie:

Wiedza i praktyka to klucz do skutecznej obrony.

Zachęcamy do aktywnego udziału w CTF-ach, poszerzania wiedzy i dzielenia się doświadczeniem z innymi.

Dziękujemy za uwagę!