Testování softwaru (KI/TSW)

Pavel Beránek

15 Dubna, 2025

9. Testování zabezpečení webu

9.1. HTML injection

HTML injection útoky jsou útoky vkládáním HTML kódu do webových stránek. Jedná se o relativně bezpečné útoky, které však mohou nadělat paseku nevhodným obsahem. Pokud je stránka slabá vůči HTML injection, tak je to příznak toho, že je rozhodně slabá na zákeřné XSS útoky.

Nejprve si spusťte následující kód, který vytvoří jednoduchou webovou stránku s přihlašovacím formulářem. Budeme potřebovat nainstalovat modul flask. Než budete dále pokračovat, tak se ujistěte, že chápete, jak webová stránka funguje. Vyzkoušejte si přidání komentáře do webové stránky.

```
from flask import Flask, request, render_template_string

app = Flask(__name__)

# In-memory "databáze" komentářů
comments = []

@app.route('/', methods=['GET', 'POST'])
def guestbook():
    if request.method == 'POST':
        comment = request.form['comment']
        comments.append(comment)

    return render_template_string('''
        <h2>Guestbook</h2>
        <p>Leave a comment - anything you write will be shown below.</p>
        <form method="POST">
            <textarea name="comment" rows="4" cols="40"></textarea><br>
            <input type="submit" value="Post Comment">
        </form>
        <hr>
        <h3>Comments:</h3>
        {% for comment in comments %}
            <div style="padding:5px; border:1px solid #ccc; margin:5px;">
                {{ comment | safe }}
            </div>
        {% endfor %}
    ''', comments=comments)

if __name__ == '__main__':
    app.run(debug=True)
```

Zadání:

1. Nejprve si otestujeme, zda je webová stránka náchylná na HTML injection útoky. Vložíme do komentáře text s HTML značkami: `<b>Tučný text</b>`. Pokud uvidíme, že se značka aplikovala na komentář, je to pro nás indikátor slabiny.
2. Zkusme udělat první útok a to na použitelnost stránky. Vložíme komentář: `<body style="background:red">`. Teď už se asi stránka nebude nikomu líbit. Pro vynulování efektu je nutné restartovat server.
3. Další slabinou a jejím jednoduchým využitím je vkládání obrázků. Můžeme vložit externí obrázek. Na fóru pro maminky nebo pro děti může být takový obrázek velký problém. Majitel webového portálu je odpovědný za obsah. `<img src="https://upload.wikimedia.org/wikipedia/commons/4/47/PNG_transparency_demonstration_1.png width="100">`
4. Hypoteticky je možné rozbít i layout. Hypoteticky jen z toho důvodu, že dnešní webové prohlížeče jsou dost odolné. Pokud je ale navázán kód na layout, tak např. takovýto kód: `<div><h1>Otevřený tag bez konce` může stránku rozbít. Z mých pokusů např. tento kód layout už rozbije: `<div><table><tr><td>`. Ještě zajímavější je vložit self-embedding iframe: `<iframe src="/" width="100%" height="100%"></iframe>`.
5. Webový portál je dále možné rozbít bombou, tedy neúměrně velkými daty. Jednou z častých možností je vložit base64 obrázek s daty (možná víte, že do jupyter lze zkopírovat byty obrázky, to je ono)> `<img src="[obrovský base64 řetězec]">`. My však provedeme něco jednoduššího a to auto-refresh hell: `<meta http-equiv="refresh" content="0">`. Dívejte se do terminálu a sledujte, co se děje.
6. Pokud někdo využívá staré verze browseru (ano, jsou firmy, které zůstali na Windows 7 a Internet Exploreru), tak je možné zatížit přímo CPU klienta pomocí marquee html prvku, který vyvolává animaci prvku měnící pozici. Nejprve si vyzkoušejte marquee jako takový: `<marquee>Tohle jede!</marquee>`. Poté uděláme útok pomocí extrémní hodnoty: `<marquee behavior="alternate" scrollamount="999999999">X</marquee>`.
7. Vymyslete si jeden zajímavý HTML útok.
8. Opravte kód webové stránky tak, aby problémy nenastaly.

Rady:

1. Nikdy nevypisujte tagy přímo, ale použijte tzv. escapování. Tagy se berou jako součást textu. Flask toto dělá rovnou, ale některé frameworky nemusí.
2. Podívejte se na na tento kód `{{ comment | safe }}`. Zjistěte účel a napravte.
3. Podívejte se na modul bleach, který je určen pro HTML sanitaci. Tím můžete vytvořit seznam povolených značek do formulářů. To je určitě lepší, než provádět úpravu vstupů tím, že odstraníte značky. To lze vždyky obejít jiným kódováním.

9.2. Cross-Site Scripting (XSS)

XSS je jeden z nejnebezpečnějších útoků na webové aplikace. Místo relativně neškodného HTML lze vložit i velmi potenciálně škodný Javascriptový kód. Ten může přesměrovat uživatele na škodlivé stránky, krást cookies a session tokeny, provádět keylogging atd. Vyjdeme z předchozí stránky pro komentáře.

1. Obdobně jako předtím začneme u jednoduchého testu zranitelnosti. Pokud následující kód projde, tak se vždy při návštěve stránky objeví alert box: `<script>alert("XSS")</script>`.
2. V alertboxu z předešlé stránky může být nějaká výzvy uživateli na přesměrování nebo vydírání (zaplacení částky nebo ...). Aby to nebylo tak nápadné pro adminy, můžeme udělat třeba to, že se objeví jen při najetí na náš komentář: `<img src="x" onmouseover="alert('hover XSS')">`.
3. Největší problém spočívá s automatickým přesměrováním, které můžeme vložit do stránky tímto způsobem: `<meta http-equiv="refresh" content="0; url=https://ki.ujep.cz/cs/">`. Teď uživatele vždy stránka přesměruje tam, kam jako útočník chceme.
4. Mnohé starší webové stránky (typicky vanilla PHP bez frameworku) řeší naivně obranu proti XSS odstraněním značek ze vstupu regulárními výrazy. To lze obejít: `<scr<script>ipt>alert('obfuscation')</scr<script>ipt>`.
5. Jedna ze starých (dnes již neúčinných technik krádeží) je získání obsahu clipboardu: `<textarea onfocus="navigator.clipboard.readText().then(t => alert(t))">Klikni</textarea>`. Neúčinná je z toho důvodu, že vyžaduje potvrzení browserem. Přesto je možné, že se nějaký jedinec nachytá, protože nečte.
6. Zkuste si vymyslet nějakou variantu XSS útoku.
7. Ošetřete web tak, aby XSS útok nebylo možné provést.

9.3. SQL Injection

SQL injection je další variantou penetračního útoku, ve které se místo HTML injektuje SQL. Jedná se tedy o útok na databázi místo webový kód. Nejprve si spusťte následující kód, který vytvoří jednoduchou webovou stránku s přihlašovacím formulářem. Než budete dále pokračovat, tak se ujistěte, že chápete, jak webová stránka funguje. Vyzkoušejte si přihlášení za uživatele a administrátora.

```
from flask import Flask, request, render_template_string, redirect
import sqlite3

app = Flask(__name__)

# Inicializace databáze
def init_db():
    conn = sqlite3.connect('users.db')
    c = conn.cursor()
    c.execute('DROP TABLE IF EXISTS users')
    c.execute('CREATE TABLE users (id INTEGER PRIMARY KEY, username TEXT, password TEXT)')
    c.execute('INSERT INTO users (username, password) VALUES ("admin", "admin123")')
    c.execute('INSERT INTO users (username, password) VALUES ("user", "user123")')
    conn.commit()
    conn.close()

init_db()

@app.route('/', methods=['GET', 'POST'])
def login():
    error = None
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']

        conn = sqlite3.connect('users.db')
        c = conn.cursor()
        query = f"SELECT * FROM users WHERE username = '{username}' AND password = '{password}'"
        print(f"DEBUG: {query}")
        c.execute(query)
        result = c.fetchone()
        conn.close()

        if result:
            return f"Welcome, {username}!"
        else:
            error = "Invalid credentials"

    return render_template_string('''
        <h2>Login</h2>
        {% if error %}<p style="color:red;">{{ error }}</p>{% endif %}
        <form method="POST">
            Username: <input type="text" name="username"><br>
            Password: <input type="text" name="password"><br>
            <input type="submit" value="Login">
        </form>
    ''', error=error)

if __name__ == '__main__':
    app.run(debug=True)
```

Zadání:

1. Přihlašte se jako admin bez znalosti hesla. Stačí zadat do username: `admin' --`. Vysvětlete na úrovni SQL, proč se povedlo.
2. Zobrazte celý obsah tabulky. To provedete například tím, že do username zadáte tautologii: `' OR 1=1 --`. Jelikož v kódu je použita metoda fetchone(), tak ji změňte na fetchall() a zjistěte, zda jste se dostali k celé tabulkce uživatelů. Vysvětlete na úrovni SQL, proč tomu tak je.
3. Pokuste se smazat databázi. To můžete provést uvedením příkazu: `'; DROP TABLE users --`. Tohle by fungovalo například v MySQL. SQLite neumožňuje vícenásobné příkazy, takže efekt neuvidíme. Zkuste alespoň vysvětlit na úrovni SQL, proč tomu tak je (že by se databáze smazala).
4. Získejte heslo prvního uživatele v databázi. Použijte k tomu příkaz do username: `' UNION SELECT 1, username, password FROM users --`. Vysvětlete, proč tomu tak je. Tento výsledek opět neuvidíte, pokud bude na stránce jen fetchone(), ale můžete ho vidět, pokud si vypíšete obsah fetchall(). Uvědomte si, že tyto útoky se mohou týkat vyhledávací kolonky, která může hledat všechny shody. U loginu se to moc neprojeví pokud to vyloženě programátor nepohňácal.
5. Vymyslete si vlastní SQL injection útok.
6. Opravte nezabezpečenou Flask aplikaci proti SQL injection útokům s využitím parametrizovaných dotazů. Vyzkoušejte, že žádný z útoku již nelze provést. Parametrizované dotazy představují hlavní nástroj ochrany proti těmto útokům. Vyhledejte si na Google termín: Python SQLite3 parametrized query.

Domácí cvičení - Útoky na website

Zkuste si vytvořit alespoň 3 vlastní útoky z každé kategorie na vlastní website. Udělejte extrémně nezabezpečenou verzi websitu a následně zabezpečenou.