VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

POKYNY PRO BEZPEČNÉ PROGRAMOVÁNÍ - REACT

SECURE CODING GUIDELINES FOR REACT

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **FILIP SOLICH**
AUTHOR

VEDOUCÍ PRÁCE **Mgr. KAMIL MALINKA, Ph.D.**
SUPERVISOR

BRNO 2022

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav inteligentních systémů (UITS) Akademický rok 2021/2022

Zadání bakalářské práce

Student: **Solich Filip**
Program: Informační technologie
Název: **Pokyny pro bezpečné programování- React**
Secure Coding Guidelines for React
Kategorie: Bezpečnost

Zadání:

1. Prostudujte problematiku bezpečného programování v Reactu.
2. Seznamte se se standardy a metodami pro bezpečné programování (např. OWASP pro webové aplikace, NIST 800-160) včetně existujících pokynů a nástrojů.
3. Navrhněte pokyny pro bezpečné programování pro React (včetně příkladů) pokrývající všechny relevantní oblasti (hlavním cílem je vytvořit kvalitní výukový nástroj). Zohledněte problematiku použitelného zabezpečení.
4. Implementujte navržený výukový nástroj a vyhodnoťte jeho použitelnost.
5. Navrhněte a implementujte reálné příklady exploitů s využitím vybraných zranitelností.

Literatura:

- https://owasp.org/
- NIST SP 800-160
- GALANSKÁ, Katarína. Usability of Usable Security Guidelines from IT Professional Point of View. Brno, 2021. Master's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Mgr. Kamil Malinka, Ph.D.

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 až 3.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Malinka Kamil, Mgr., Ph.D.**
Vedoucí ústavu: Hanáček Petr, doc. Dr. Ing.
Datum zadání: 1. listopadu 2021
Datum odevzdání: 11. května 2022
Datum schválení: 3. listopadu 2021

Abstrakt

Tato práce se zabývá psaním bezpečných aplikací v JavaScriptové knihovně React. Cílem této práce je vytvořit příručku pro programátory, aby s její pomocí byli schopni detekovat části webových aplikací, které mohou být zneužité k útoku na aplikaci. Je zde popsáno jak a na co je třeba si pří psaní webových aplikací dát pozor, jaké jsou nejlepší praktiky programování v knihovně React, díky kterým se programátor dokáže vyhnout bezpečnostním chybám v kódu aplikace a jak případné chyby opravit. Jsou zde popsány i samotné druhy útoků a jak mohou útoky na zranitelnou aplikaci probíhat. Znalost průběhu útoku pomůže programátorovi lépe přemýšlet o slabých článcích aplikace a tím také odhalit bezpečnostní problém v aplikaci dříve než útočník.

Abstract

This work deals with writing secure applications in JavaScript library React. The aim of this work is to create a guide for programmers to be able to detect parts of web applications that can be exploited to attack on the application. It describes how and to what you need to pay attention to when writing web applications, what are the best programming practices in the React library, thanks to which the programmer can avoid security errors in the application code and how to fix any errors. The types of attacks themselves and how attacks on a vulnerable application can take place are also described here. Knowing the progress of the attack will help the programmer to think better about the weak links of the application and thus also detect a security issue in the application before the attacker.

Klíčová slova

ReactJS, JavaScript, TSX, TypeScript, OWASP, Bezpečnost, XSS, CSRF, API, NIST 800-160, Webové aplikace

Keywords

ReactJS, JavaScript, TSX, TypeScript, OWASP, Security, XSS, CSRF, API, NIST 800-160, Web applications

Citace

SOLICH, Filip. *Pokyny pro bezpečné programování - React*. Brno, 2022. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Mgr. Kamil Malinka, Ph.D.

Pokyny pro bezpečné programování - React

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Mgr. Kamila Malinky, Ph.D. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Filip Solich
10. května 2022

Poděkování

Chtěl bych poděkovat vedoucímu mé práce Mgr. Kamilu Malinkovi, Ph.D. za odborné vedení při tvorbě této práce. Dále bych chtěl poděkovat Ing. Maroši Barabasovi, Ph.D. za odborné konzultace.

Obsah

Kapitola 1

Úvod

Bezpečnost dnešních aplikací se pořád zlepšuje. Stejně tak se ale zlepšují techniky útoku na tyto aplikace, a zejména na webové aplikace, protože ty jsou v dnešní době jedny z nejrozšířenějších a z velké části se logika těchto aplikací provádí přímo na zařízení uživatele. Toto je umožněno stažením zdrojových kódu v jazyce JavaScript. Takto jakýkoli útočník může kód prostudovat a navrhnout vhodný útok na aplikaci.

Aby byly aplikace co nejbezpečnější, musí programátoři vědět, jak bezpečné aplikace psát, co může způsobit zranitelnost aplikace a jak se ji vyhnout. Tato práce tedy slouží i jako příručka programátorům. Práce je zaměřena hlavně na bezpečné praktiky programování za použití knihovny React[1] pro jazyk JavaScript.

Ve druhé kapitole je seznam použitých technologií a vysvětlení základů nutných pro pochopení práce. Je zde vysvětleno použití JavaScriptu a knihovny React, která je v JavaScriptu napsaná. Dále je zde nastínění některých syntaktických vylepšení, které lze s knihovnou React použít.

Třetí kapitola pojednává o bezpečném programování aplikací za pomocí knihovny React. Nejdříve jsou v této kapitole popsány bezpečnostní problémy jazyka JavaScript, což je také jeden z důvodů, proč si musí programátor dávat při psaní webových aplikací pozor. Dále jsou v této kapitole popsány nejdůležitější zranitelnosti JavaScript aplikací a jakou obranu proti těmto zranitelnostem React poskytuje, nebo jak se proti nim může ubránit programátor při své práci. A dále zde jsou popsány správné praktiky programovaní, které když programátor dodrží, tak se může vyhnout některým zranitelnostem.

Ve čtvrté kapitole jsou představena již existující řešení a programy pro problematiku bezpečnosti aplikací, která dokáží programátorovi pomoci při vytváření bezpečných aplikací.

Pátá kapitola se zabývá návrhem aplikace, které slouží jako příručka pro programátoru programující v Reactu. Je zde popsán návrh samotné aplikace a návrh databáze.

V kapitole číslo 6 je popsána implementace této příručky. Popis obsahuje vše od výběru technologií přes tvorbu aplikace až po nasazení aplikace na server.

Sedmá kapitola je zaměřena na implementaci aplikace, která slouží jako ukázka útoku na zranitelnou aplikaci, která využívá starou verzi knihovny React. Tato aplikace je naprosto oddělená od příručky.

V osmé kapitole je poté popsáno testování obou aplikací. Testovala se převážně funkčnost obou aplikací.

[1] https://reactjs.org/

Kapitola 2

Popis použitých technologií

V této kapitole jsou stručně vysvětleny technologie, které musí programátor ovládat, aby byl schopen s knihovnou React pracovat. Je zde popsán i důvod, proč knihovnu React použít a co jsou její přednosti.

2.1 JavaScript

JavaScript je programovací jazyk používaný pro psaní skriptů, které jsou spouštěny v prohlížeči uživatele webové aplikace. JavaScript se používá hlavně pro změnu DOM prvků webové stránky bez nutnosti překreslování celé stránky. JavaScript podporuje asynchronní programování, díky kterému může program stáhnout data ze serveru na pozadí aplikace. Poté co se data stáhnou, může program překreslit konkrétní prvky stránky opět bez nutnosti překreslování celého dokumentu. Zdrojové kódy jazyka JavaScript mají standardně příponu `.js`.

Díky JavaScriptu je možné snížit hardwarové nároky serveru, protože lze některé operace a výpočty provést přímo na počítači uživatele.

2.2 React

React je JavaScriptová knihovna vyvíjena firmou Meta od roku 2013[22]. Knihovna se používá pro programování frontendové části webových aplikací a v roce 2022 patří spolu s Vue.js[1] a Angular[2] k nejpoužívanějším JavaScriptovým knihovnám.

Struktura HTML dokumentu se v Reactu vytváří pomocí programátorem vytvořených komponent. Tyto komponenty mohou být vytvořeny jako obyčejné JavaScriptové funkce, které vracejí objekt typu React komponent, nebo jako JavaScript třída, která dědí z `React.Component` třídy a implementuje metodu `render`, která vrací objekt typu React komponent.

Knihovna React sama o sobě neobsahuje tolik funkcionalit a spoléhá na to, že všechny potřebné funkcionality si formou knihoven vytvoří komunita vývojářů používajících React. Takto kolem Reactu vzniká ekosystém knihoven poskytujících vše, co by mohl programátor při vytváření webové aplikace potřebovat. Jmenovitě například knihovna Gatsby[5] pro vytváření statických webových stránek, nebo knihovna Spring[10] pro tvorbu animací.

[1] https://v3.vuejs.org/
[2] https://angular.io/

2.3 JSX

Tato sekce je převzata z dokumentace knihovny React, konkrétně kapitoly Introducing JSX[20] a JSX In Depth[21].

JSX je syntaktické rozšíření jazyka JavaScript. Při vytváření aplikace pomocí knihovny React je doporučeno používat právě syntaxi JSX pro ulehčení práce. Zdrojové soubory obsahující kód psaný v této syntaxi mívají příponu `.jsx`. JSX kombinuje HTML elementy s JavaScriptovým kódem, jak lze vidět na ukázce níže.

```
// Ukazka syntaktickeho rozsireni JSX

// Element vyvoreny pomoci JSX
const element = <h1>Hello, world!</h1>;

// Stejny element vytvoreny ciste pomoci JavaScriptu
const element = React.createElement('h1', null, 'Hello, world!');
```

Další výhodou používaní JSX při psaní aplikací pomocí knihovny React je možnost vložení výrazu přímo do JSX elementu. Výrazem může být jakýkoli validní kus kódu v jazyce JavaScript. Výrazem může být také volání funkce, která vrací JSX element.

```
const name = 'Josh Perez';

const element = <h1>Hello, {name}</h1>; // Element obsahuje text "Hello, Josh Perez"
```

Na tom, jestli programátor bude aplikaci psát pomocí JSX nebo elementy vytvářet pomocí `React.createElement`, nezáleží. Internetový prohlížeč skriptu napsanému pomocí JSX nerozumí, a proto se každý takto napsaný soubor musí zkompilovat do čistého JavaScript kódu využívajícího funkci `React.createElement`. Název React komponenty v JSX musí vždy začínat velkým písmenem, v opačném případě bude React komponentu považovat za HTML element a aplikace nebude fungovat.

2.4 TypeScript

TypeScript je nadmnožina jazyka JavaScript a doplňuje tento jazyk o definici datových typů. Díky tomu lze odhalit chyby v programu, související se špatnou prací s datovými typy, už během psaní programu, a ne až za běhu aplikace, jak je tomu u JavaScriptu. Zdrojové kódy jazyka TypeScript mají standardně příponu `.ts`. Internetový prohlížeč jazyku TypeScript nerozumí, stejně jako tomu je u syntaxe JSX, proto se musí všechny zdrojové kódy napsané v jazyce TypeScript přeložit do jazyka JavaScript. Tento překlad se provádí programem `tsc`[23]. Konfigurace chování programu `tsc` se provádí v souboru `tsconfig.json`. Zdrojový kód doplněný o datové typy se také stává sebedokumentujícím, čehož můžou využít našeptávače vývojových prostředí.

Používání TypeScriptu pomáhá předcházet chybám, a díky tomu můžou být aplikace v něm napsané bezpečnější. Podle firmy Airbnb by se použitím TypeScriptu dalo předejít až 38 % chyb v jejich programech[16].

Kapitola 3

Pokyny pro bezpečné programování

V této kapitole jsou v jednotlivých sekcích popsány zranitelnosti, které mohou vzniknout v programech psaných za pomocí knihovny React, a jaká bezpečnostní rizika s sebou nese samotný programovací jazyk JavaScript. Také jsou zde popsány příklady, jak se těmto zranitelnostem vyhnout a na co je třeba si dát pozor, tak aby mohl programátor psát bezpečné aplikace.

3.1 Bezpečností problémy jazyka JavaScript

Přestože je JavaScript užitečný programovací jazyk, který je v dnešní době součástí většiny webových aplikací, dokáže být také velice nebezpečným jazykem právě díky schopnosti spouštět skripty stažené internetovým prohlížečem z různých zdrojů.

Otisk prohlížeče

Otisku prohlížeče se anglicky říká fingerprinting. Otisk prohlížeče slouží k identifikaci uživatele. Pomocí otisku lze identifikovat i uživatele, který není přihlášen, a na jehož zařízení nebyl vytvořen žádný soubor cookie. Otisk prohlížeče může sloužit firmě pro rozpoznání, o jakého uživatele se jedná, a podle toho mu zobrazit cílenou reklamu, anebo taky může sloužit útočníkovi pro sledováni oběti napříč weby. Strana, která se otisk snaží provést, to udělá tak, že se o daném zařízení pokusí získat co nejvíce informací a ty odešle na svůj server pro další zpracování. Informace, které se pro otisk dají získat, vypadají samy o sobě bezvýznamně, ale pokud je těchto informací dostatek, mohou pomoci jednoznačně odhalit, kdo je uživatelem. Pro to, aby webová aplikace fungovala nebo aby bylo možné jí optimalizovat pro různá zařízení, má k těmto informacím přístup JavaScript. Mezi takové informace patří:

- datum a čas, časová zóna
- název a verze internetového prohlížeče
- operační systém zařízení
- velikost okna, velikost dokumentu, barevná hloubka
- IP adresa zařízení

- připojená zařízení (audio, video)
- geolokační souřadnice*
- ...

*Přístup k těmto datům musí povolit uživatel explicitně.

Přístup k souborovému systému

Pro zachování bezpečnosti JavaScript nemá přístup do souborového systému, to znamená že nemůže číst nebo zapisovat soubory na disk uživatele. Pokud chce uživatel stáhnout soubor (například klikne na HTML odkaz, který má nastaven atribut `download` s hodnotou, která udává, jak se stahovaný soubor bude jmenovat), tak prohlížeč vytvoří okno, ve kterém musí uživatel stažení odsouhlasit. Pokud JavaScript potřebuje přečíst nějaký soubor, tak ho musí uživatel sám vybrat a vložit do HTML formulářového vstupu `<input type="file"/>`.

3.2 Knihovny třetích stran

Při programování se často stává, že chceme naprogramovat něco, co už naprogramovala spousta lidí před námi. Programátoři jsou si toho dobře vědomi a tak se někdy rozhodnou vydat svůj kód jako knihovnu, aby ostatním ulehčili práci. Takové knihovny mají často otevřený zdrojový kód, takže je může kdokoli upravit. Pokud na knihovně začne pracovat více programátorů, je větší šance, že kód v knihovně bude rychlejší, bezpečnější a bude obsahovat více funkcionalit. Použití knihovny tedy vede nejen k urychlení naší práce, protože nemusíme „znovu vynalézat kolo“, ale také může vést k vytváření kvalitnějších aplikací.

V mnoha programovacích jazycích se pro stahování knihoven používá takzvaný správce balíčku (package manager). V JavaScriptu jsou oblíbení dva správci - npm[1] a Yarn[2]. Autor knihovny, vždy když vydá novou verzi své knihovny, ji nahraje na některý, případně i na všechny správce balíčků, a tím svou knihovnu zpřístupní programátorům. Ti si poté mohou knihovnu jednoduše stáhnout nebo aktualizovat na nejnovější verzi. Stahování obvykle probíhá přes příkazový řádek. Stažení knihovny React přes správce balíčků se provádí nasledujícími příkazy:

- npm - `npm i react`
- Yarn - `yarn add react`

Tyto knihovny ale mohou obsahovat zranitelnosti nebo mohou samy používat jiné knihovny, které obsahují zranitelnosti. Na to je třeba při volbě knihovny vždy myslet a někdy proto může být lepší se obejít zcela bez cizí knihovny a řešení si udělat po svém. Při výběru správné knihovny a rozhodování, jestli neexistuje lepší alternativa, je dobré zohlednit následující body.

- Knihovna je nejvíce používaná jinými programátory a odpovídá našim potřebám
- Knihovna je stále ve vývoji

[1] https://www.npmjs.com
[2] https://yarnpkg.com

- Knihovna je ve verzi vyšší než 1.0

Tyto body samy o sobě neurčují, zda je knihovna bezpečná, ale jedná se o dobré indikátory toho, v jakém stavu knihovna je. Pokud se najde nová zranitelnost v existující knihovně, která je stále ve vývoji, tak je zde šance, že autoři tuto zranitelnost opraví a opravu zahrnou do další verze. Při vývoji aplikací, kde je bezpečí nejvyšší prioritou, je dobrý nápad zkontrolovat i zdrojové kódy knihovny. Čím více lidí si zdrojové kódy přečte, tím vyšší je šance, že v nich někdo objeví chybu.

Nakonec ať už používáme jakoukoli knihovnu, je třeba ji pravidelně aktualizovat na nejnovější verzi obsahující bezpečnostní záplaty. Pro jednoduché zjištění, které balíčky obsahují zranitelnosti, a zda je lze aktualizovat, slouží příkaz `audit`, který podporují oba správci balíčků, kteří již byli zmíněni. Spouští se příkazy:

- npm - `npm audit`
- Yarn - `yarn audit`

Pro instalaci specifické verze knihovny lze za název knihovny přidat znak `@` a číslo verze, kterou chceme stánout.

- npm - `npm i react@18.1.0`
- Yarn - `yarn add react@18.1.0`

3.3 Vložení citlivých informací do zdrojového kódu

Jednou z nevýhod programovacího jazyku JavaScript a značkovacího jazyku HTML je to, že jsou distribuovány ve své zdrojové podobě. To znamená, že jakmile uživatel přistoupí na naši webovou aplikaci, se mu do internetového prohlížeče stáhnou zdrojové kódy. Internetový prohlížeč poté zdrojové kódy přečte. Vzhled a text webové stránky se vykreslí z HTML kódu a skript napsaný v JavaScriptu spustí interpret, který je součástí internetového prohlížeče. Z tohoto důvodu je třeba si dát pozor aby naše zdrojové kódy neobsahovaly tajné informace, ke kterým by se neměl nikdo dostat. Mezi takové informace patří:

- Přihlašovací jména
- Hesla
- Adresy k dalším serverům, se kterými může aplikace komunikovat
- API klíče

S kontrolou, zda programátorovi neunikly citlivé informace, může pomoci řešení od firmy GitGuardian4.3 nebo GitHub4.4.

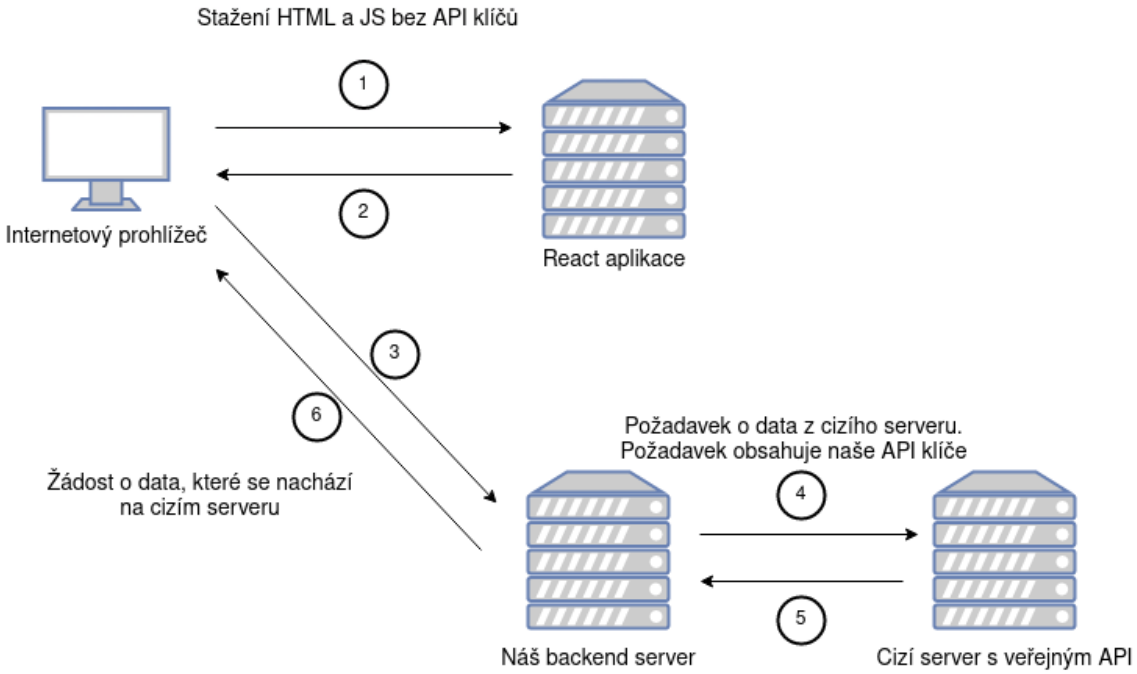
3.3.1 Co jsou API klíče a proč se považují za citlivé informace

Naše aplikace někdy potřebuje přistoupit k informacím z jiných zdrojů. Tato komunikace probíhá přes rozhraní pro programování aplikací (anglicky API - Application Programming Interface). Některé služby poskytující veřejné API vyžadují, abychom si u nich nejprve vytvořili účet a zaregistrovali naši aplikaci. Toto vetšinou vyžadují služby, které mají přístup

ke svému API placený. Registrací získáme identifikátor pro naši aplikaci a tajný klíč. Pokud chceme získat údaje z API, při každém HTTP požadavku na API musíme spolu s požadavkem vždy zaslat identifikátor naší aplikace a tajný klíč. Tyto informace jsou pro správce API důležité, aby věděl, z jaké aplikace požadavek přišel. Služba si poté může účtovat cenu za využívání jeho API například podle počtu HTTP požadavků za měsíc.

Kdyby HTTP požadavky na takovéhoto API posílal přímo JavaScriptový kód z prohlížeče klienta, musel by mít přístup k identifikátoru a soukromému klíči naší aplikace. To by znamenalo že by tyto údaje musely být součástí HTML nebo JavaScriptového kódu, a tedy by si je mohl přečíst každý uživatel naší aplikace. Velice jednoduše by se tedy k identifikátoru aplikace a tajnému klíči dostal útočník, který by tyto dva údaje mohl zneužít k posílání HTTP požadavků na API službu, kterou využíváme. Požadavky od útočníka by díky těmto klíčům vypadaly, jako by přišly z naší aplikace. Takto by útočník mohl třeba vypotřebovat náš měsíční předplacený limit a naše aplikace by byla nadále nefunkční. Z toho důvodu je potřeba identifikátor aplikace a tajný klíč uchovat v naprostém bezpečí a zacházet s těmito údaji podobně, jako s hesly.

Jedním z možných způsobů, jak citlivé klíče ochránit, je spustit si vlastní API na svém backend serveru. React aplikace poté bude posílat HTTP požadavky na API na backend serveru, a ten pošle HTTP požadavek s API klíči na API službu, se kterou chceme komunikovat.



Obrázek 3.1: Bezpečná komunikace s cizím serverem, který vyžaduje API klíče

3.4 Skriptování napříč weby

Anglicky se tento druh zranitelnosti nazývá cross-site scripting, zkráceně XSS. Tato zranitelnost je jednou z nejnebezpečnějších, protože umožňuje spouštět nebezpečný JavaScriptový kód v internetovém prohlížeči oběti.

Zranitelnost může vzniknout, pokud programátor vykreslí do HTML stránky kód, o kterém neví, co obsahuje (například HTML kód z nějaké textové proměnné). Pokud takový text obsahuje JavaScriptový kód a není před vykreslením nijak ošetřen, tak místo toho, aby se skript vypsal jako text na obrazovku, se může kód spustit. Co vše může nebezpečný skript udělat, je popsáno v sekci Bezpečnostní problémy jazyka JavaScript3.1.

Jak se bránit proti XSS

Následující sekce je převzata z příručky OWASP[26].

Obrana proti XSS se liší podle toho, na které straně se provádí, jestli na serverové straně aplikace nebo na klientské straně.

- Reflected or Stored - Ochrana se provádí na serveru
- DOM Based XSS - Ochrana se provádí na klientovi

Obrana proti XSS založenému na DOM

Následující sekce je převzata z příručky OWASP DOM based XSS Prevention[19].

Pokud je útok založen na objektovém modelu dokumentu (anglicky Document Object Model neboli DOM), znamená to, že škodlivý kód je do stránky vložen přímo za běhu aplikace na straně klienta.

1. Pravidlo V JavaScriptovém kódu je třeba ošetřit všechen HTML kód a potom všechen JavaScriptový kód, o kterém nevíme, co obsahuje, anebo když pochází z neznámých zdrojů, předtím, než je vložen do HTML. Pro toto ošetření lze použít třeba JavaScriptovou knihovnu `node-esapi`[17].

2. Pravidlo V JavaScriptovém kódu je třeba ošetřit všechen JavaScriptový kód, o kterém nevíme, co obsahuje, anebo když pochází z neznámých zdrojů, předtím, než je vložen do atributu HTML elementu.

XSS v knihovně React

Pokud je knihovna React použita se syntaktickým dodatkem JSX, vždy se postará o to, že obsah proměnné, který se programátor chystá vykreslit, je před vykreslením upraven tak, aby výsledkem nebyl spustitelný kód[3]. Tento bezpečnostní prvek je ale někdy potřeba obejít - pokud například potřebujeme stáhnout skript ze stránky, kterou zadá uživatel, a jsme si jistí, že daná stránka je bezpečná. Knihovna React má pro tyto případy zvláštní atribut `dangerouslySetInnerHTML`. Tomuto atributu je ještě třeba předat objekt s klíčem `__html`. Syntaxe je záměrně dlouhá a ošklivá, aby programátorovi při psaní připomněla, že dělá něco nebezpečného, a že by to dělat neměl, pokud si není jistý tím, co dělá[4].

```
// !!! Ukazka nebezpecneho kodu !!!
var variable // Promenna obsahujici nebezpecny kod

function MyComponent() {
  return <div dangerouslySetInnerHTML={{__html: variable}} />;
}
```

[3] https://reactjs.org/docs/introducing-jsx.html#jsx-prevents-injection-attacks
[4] https://reactjs.org/docs/dom-elements.html#dangerouslysetinnerhtml

Špatné a nebezpečné použití atributu `dangerouslySetInnerHTML` je vidět ve staré desktopové verzi aplikace Signal, která slouží pro bezpečnou a soukromou komunikaci[30]. V aplikaci Signal bylo `dangerouslySetInnerHTML` použito pro vykreslení přijatých zpráv. Díky této chybě stačilo útočníkovi, aby poslal oběti nebezpečnou zprávu obsahující skript. Tato chyba již byla opravena[24].

JavaScriptový kód lze vložit také do URL. To znamená, že je třeba ověřovat všechny URL, které byly zadány uživatelem nebo staženy z externích zdrojů. Jednou z možností jak zjistit, zda URL adresa neobsahuje skript, je podívat se na URL protokol. Bezpečný protokol je `http`, nebo `https`. Tohoto lze docílit pomocí kódu níže[28]:

```
// Ukazka validace URL
// Autor: Ron Perris
function validateURL(url) {
  const parsed = new URL(url)
  return ['https:', 'http:'].includes(parsed.protocol)
}

<a href={validateURL(url) ? url : ''}>Click here!</a>
```

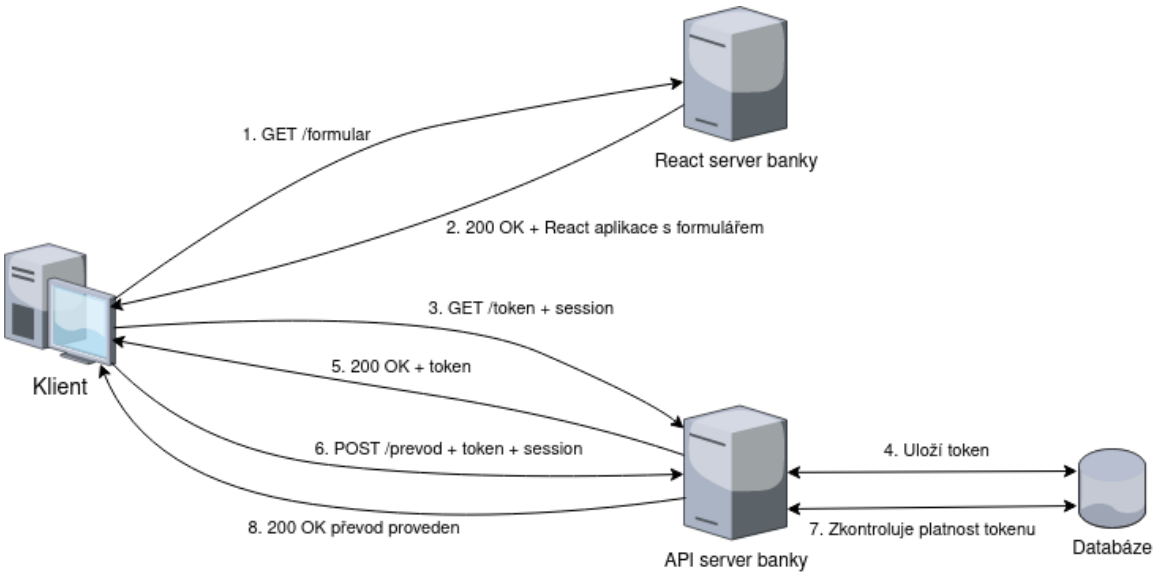
3.5 Falšování požadavků napříč weby

Anglicky se tato zranitelnost nazývá cross-site request forgery (zkratka CSRF). Díky tomuto typu útkou je možné, aby stránka útočníka provedla HTTP požadavek na stránku, na kterou se útočník snaží útočit. Zároveň se tento požadavek tváří, jako by pocházel od uživatele, který je přihlášen na napadené stránce a navštívil stránku útočníka.

Pro příklad si představme, že používáme internetové bankovnictví. Do internetového bankovnictví jsme přihlášeni, takže je v našem prohlížeči uloženo sezení (anglicky session), které se odesílá jakou součást každého HTTP požadavku, který na stránku internetového bankovnictví odešleme. Nějaký útočník se nás snaží obrat o peníze a ví, jakou banku používáme. Této znalosti může využít a pomocí zpětného inženýrství internetového bankovnictví zjistí, že pokud chce klient banky provést převod peněz, tak vyplní formulář, ve kterém zadá údaje na jaký účet chce převod provést a kolik peněz chce převést. Poté co klient formulář vyplní a stiskne tlačítko Odeslat, tak se formulář odešle na URL adresu `https://www.banka.cz/prevod`. Útočník vytvoří svou webovou stránku, na kterou pošle oběti odkaz třeba přes email. Oběť na odkaz klikne, protože stránka podle URL adresy nevypadá nebezpečně. Tato stránka bude obsahovat stejný formulář, který obsahuje stránka internetového bankovnictví, ale všechny položky formuláře udělá skrytě pomocí `<input type="hidden"... />` tak, aby si oběť ničeho nevšimla. V tomto formuláři bude číslo účtu příjemce převodu předvyplněno na číslo účtu útočníka a předvyplněna bude i částka převodu. Jakmile se komukoli tato webová stránka načte, tak se na pozadí spustí JavaScriptový kód, který formulář odešle na URL adresu `https://www.banka.cz/prevod`. Takto se spolu s formulářem odešle i session uživatele, který přistoupil na útočníkovu stránku. Banka si bude díky platné session myslet, že dotaz na provedení převodu je validní, a tedy jej uskuteční.

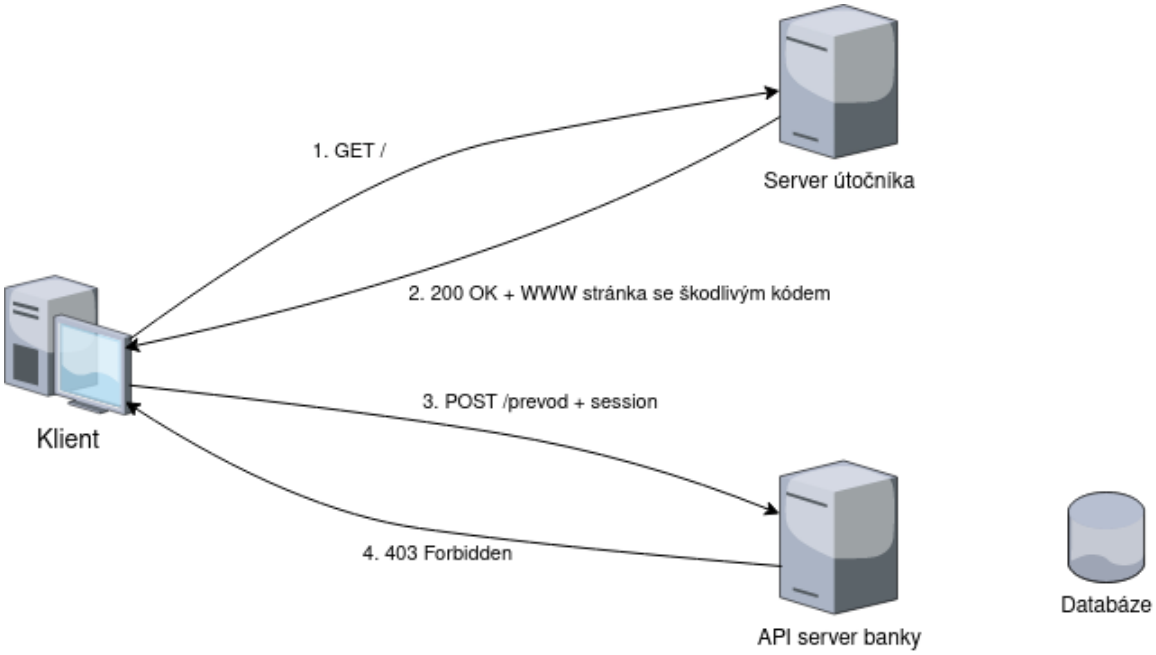
Tomuto typu útoku lze předejít pomocí serverem vygenerovaného tokenu, který musí být součástí požadavku pro převod peněz. Jako token se obvykle používá náhodný řetězec tvořen ASCII znaky. Po vygenerování si server musí token uložit, protože byl vygenerován konkrétnímu uživateli a má jen omezenou dobu platnosti. Pokud je stránka s formulářem sestavena na serveru, tak server formulář sestaví rovnou s tokenem. V případě React aplikací

se často používá metoda, kdy je stránka sestavena až na klientské straně, a proto musí aplikace získat token přes domluvené API. V praxi to znamená, že React aplikace pošle dotaz například na URL adresu `https://banka.cz/token`. Server vygeneruje token, který asociuje se session dotazu, uloží jej do databáze a pošle zpátky React aplikaci. Ta jej přidá do formuláře, aby se spolu s ním odeslal. Server poté zkontroluje, zda má daný token v databázi, a zda je validní.



Obrázek 3.2: Komunikace se servery banky při provádění převodu peněz

V případě, kdy oběť přistoupí na stránky útočníka a spustí se skript, který odešle požadavek na provedení platby z účtu oběti, tento požadavek nebude obsahovat token, a tudíž jej server banky vyhodnotí jako neplatný. Útočník nemůže získat token z adresy `https://banka.cz/token`, protože server si předtím, než token vygeneruje, zkontroluje, odkud požadavek pochází. Server banky vygeneruje token, jen pokud požadavek na vygenerování pochází ze stránek banky.



Obrázek 3.3: Komunikace, pokud oběť přistoupí na server útočníka

3.6 Špatně implementovaná autentizace (Broken Authentication)

Autentizace je proces při kterém uživatel používající aplikaci prokáže, že je skutečně osobou, za kterou se vydává (například se snaží přihlásit ke konkrétnímu uživatelskému účtu, který je v aplikaci registrovaný). Tento proces je z pohledu zabezpečení velice složitý a musí být implementován bezchybně. Chyba v implementaci autentizace může vést ke krádeži účtů a odcizení citlivých informací uživatele.

3.6.1 Typy autentizace

Autentizovat uživatele lze několika způsoby. Nejčastěji používaný způsob je pomocí dvojice textových řetězců, kde jeden slouží k identifikaci účtu, ke kterému se uživatel snaží přihlásit (jméno, login, email, ...), a ten druhý je heslo, které zná pouze majitel účtu. Přestože tento způsob je nejpoužívanější, není to zdaleka jediný způsob autentizace. Další způsoby jsou uvedeny v seznamu níže.

- Přihlášení pomocí aktivního sezení (session)
- Autentizace u aplikace 3 strany
- Bezheslová autentizace pomocí aplikace v telefonu
- Bezheslová autentizace pomocí hardwarového klíče

3.6.2 Sezení

HTTP je bezstavový protokol, to znamená že webserver si nepamatuje s jakým uživatelem komunikuje, a ten proto musí dokázat svou totožnost při odeslaní každého HTTP poža-

davku. Vyplňovat před odesláním každého požadavku přihlašovací jméno a heslo by bylo velice nepraktické, proto jakmile se uživatel jednou přihlásí, webserver vytvoří pro daného uživatele sezení. Sezení je unikátní identifikátor, který vygeneruje webserver a pošle ho v HTTP odpovědi poté, co se uživatel úspěšně přihlásil. Webový prohlížeč tento identifikátor uloží v podobě HTTP cookie (český překlad je "sušenka", ten se ale téměř vůbec nepoužívá). HTTP cookie je malý blok dat poslaný webserverem v HTTP odpovědi. Webový prohlížeč tato data uloží a pamatuje si, se kterou doménou jsou data asociována. Pokaždé, když webový prohlížeč sestavuje nový HTTP požadavek, k němu přibalí všechny cookies, které má uloženy a asociovány s doménou, na kterou HTTP požadavek směřuje. Server si ID sezení v příchozím HTTP požadavku spojí s konkrétním uživatelem, pro kterého toto ID vygeneroval, a ví tedy, od koho tento HTTP požadavek přichází.

Pokud by se útočník dostal k ID sezení, mohl by se vydávat za uživatele, pro kterého byl tento identifikátor vygenerován.

3.6.3 Chyby v autentizaci

Různých chyb může v autentizaci vniknout několik. Některé z nich, které souvisí s použitím Reactu nebo jiné frontendové knihovny, jsou popsány v následujících odstavcích.

Nebezpečné spojení se serverem

Pokud spojení se serverem není zabezpečeno přes HTTPS, lze odcizit přihlašovací údaje z HTTP požadavku. Tyto údaje lze odcizit pomocí útoku typu man-in-the-middle, při kterém útočník odposlouchává naši veškerou komunikaci.[5]

ID sezení v URL adrese

Jak je popsáno v sekci 3.6.2, odcizení identifikátoru sezení by mělo za následek, že by se mohl útočník, který tento identifikátor odcizil, vydávat za někoho jiného, a web server by to nepoznal. Je tedy třeba s identifikátorem sezení zacházet opatrně. Je důležité si rozmyslet, jak sezení přidat k HTTP požadavku.

Bezpečný způsob jak toho docílit, je nechat webový prohlížeč, ať sezení dá do hlavičky HTTP požadavku spolu s ostatními cookies. Toto je výchozí chování webových prohlížečů, pokud byl identifikátor sezení uložen jako cookie.

Problém vniká, když se programátor rozhodne přidat identifikátor jako část URL adresy. Příklad vložení identifikátoru sezeni do URL adresy. "xxx" je v těchto případech identifikátor.

- `https://domena.cz/uzivatel/xxx` - vložení identifikátoru do URL cesty
- `https://domena.cz/uzivatel?session=xxx` - vložení identifikátoru jako parametr metody GET

V obou případech je hodnota sezení lehce čitelná z URL. Útočník by tuto hodnotu mohl přečíst, pokud by uživatel aplikaci používal někde na veřejném místě.

[5] https://cs.wikipedia.org/wiki/Man_in_the_middle

Dlouhá životnost sezení

Životnost sezení lze omezit. Pokud je sezení uchováváno jako sušenka v prohlížeči, webserver při nastavování sezení může určit jeho životnost. Čím kratší životnost sezení, tím menší je šance, že jej zneužije útočník. Uživatelé ale chtějí aplikaci jednoduše používat bez nutnosti zadávat své heslo každou hodinu. Je proto vhodné nastavit životnost sezení na dobu mezi 1 a 3 měsíci. Dále je třeba sezení odstranit pokaždé, když se uživatel odhlásí.

Částečné vyzrazení přihlašovacích údajů

Pokud uživatel zadá při přihlášení špatně jméno nebo heslo, musí být o své chybě informován. V praxi se toto řeší nejčastěji chybovou hláškou s textem, vysvětlujícím proč se uživatel nemůže přihlásit. Tato informace ale nesmí být příliš napovídající, aby ji nemohl útočník zneužít.

Pokud útočník hádá náhodné kombinace třeba emailu a hesla, je potřeba při špatné kombinaci napsat hlášku ve stylu "špatně zadaný email nebo heslo". Kdyby chybová hláška byla moc informativní a napsala třeba jen "špatné heslo", tak by útočníkovi napověděla, že v aplikaci je registrován účet s emailem, který útočník zkoušel. Poté by útočník už nezkoušel náhodné kombinace email a heslo, ale zkoušel by jen různé hesla pro email, o kterém ví, že je spojen s existujícím účtem. To by útočníkovi značně ulehčilo práci.

I když se útočník nesnaží neoprávněně se zmocnit cizího účtu, stejně může informace o existujícím účtu zneužít. Pokud chce útočník provést útok na konkrétní osobu, hodí se mu znát co nejvíce informací o dané osobě. Pokud tedy zná email své oběti útoku, může zjistit, jaké aplikace a služby daná osoba používá tím, že bude zkoušet zadávat její email do různých aplikací a z chybových hlášek vyvozovat, zda daná osoba službu používá.

Kapitola 4

Existující řešení

Tato kapitola popisuje existující programy, které pomáhají programátorům psát bezpečný kód.

4.1 Open Web Application Security Project

Zkráceně OWASP je nezisková organizace založená roku 2001[25]. Tato organizace se zabývá pořádáním konferencí a přednášek na téma bezpečnosti webových aplikací. Mimo jiné má organizace na starost projekty zaměřující se na bezpečnost v různých odvětvích aplikací.

OWASP top 10

OWASP top 10 je jeden z nejznámějších projektů organizace, a to z toho důvodu, že jednoduše a přehledně popisuje 10 nejčastěji používaných zranitelností ve webových aplikacích[27]. Seznam byl naposled aktualizován v roce 2021. Těmito 10 pravidly by se měl určitě řídit vývoj každé webové aplikace, tak aby byla aplikace zabezpečena alespoň proti nejběžnějším typům útoků. Tento seznam samozřejmě neobsahuje všechny druhy zranitelností webových aplikací, a tedy pokud je aplikace zabezpečena proti všem možným útokům z tohoto seznamu, ještě to neznamená, že je aplikace perfektně zabezpečená. Seznam zahrnuje typy zranitelností jak frontendové, tak i backendové.

4.2 NIST 800-160

Definice je převzata a přeložena z webu nist.gov[12]

NIST je zkratka pro National Institute of Standards and Technology. Tento název přeložen do češtiny znamená Národní Institut Standardů a Technologie a patří pod ministerstvo obchodu Spojených států amerických. Úkolem této instituce je podpora průmyslové konkurenceschopnosti zlepšováním vědeckých měření, standardů a technologií.

NIST 800-160 je dokument vydáván organizací NIST a jeho celý název je Developing Cyber-Resilient Systems (Vývoj kyberneticky odolných systémů). Tento dokument lze brát jako příručku pro dosažení kybernetické bezpečnosti sytému, spolu s růstem velikosti systému. Dále dokument popisuje konstrukce kybernetické bezpečnosti, jako jsou cíle, techniky a přístupy pro navržení bezpečných systémů.[29]

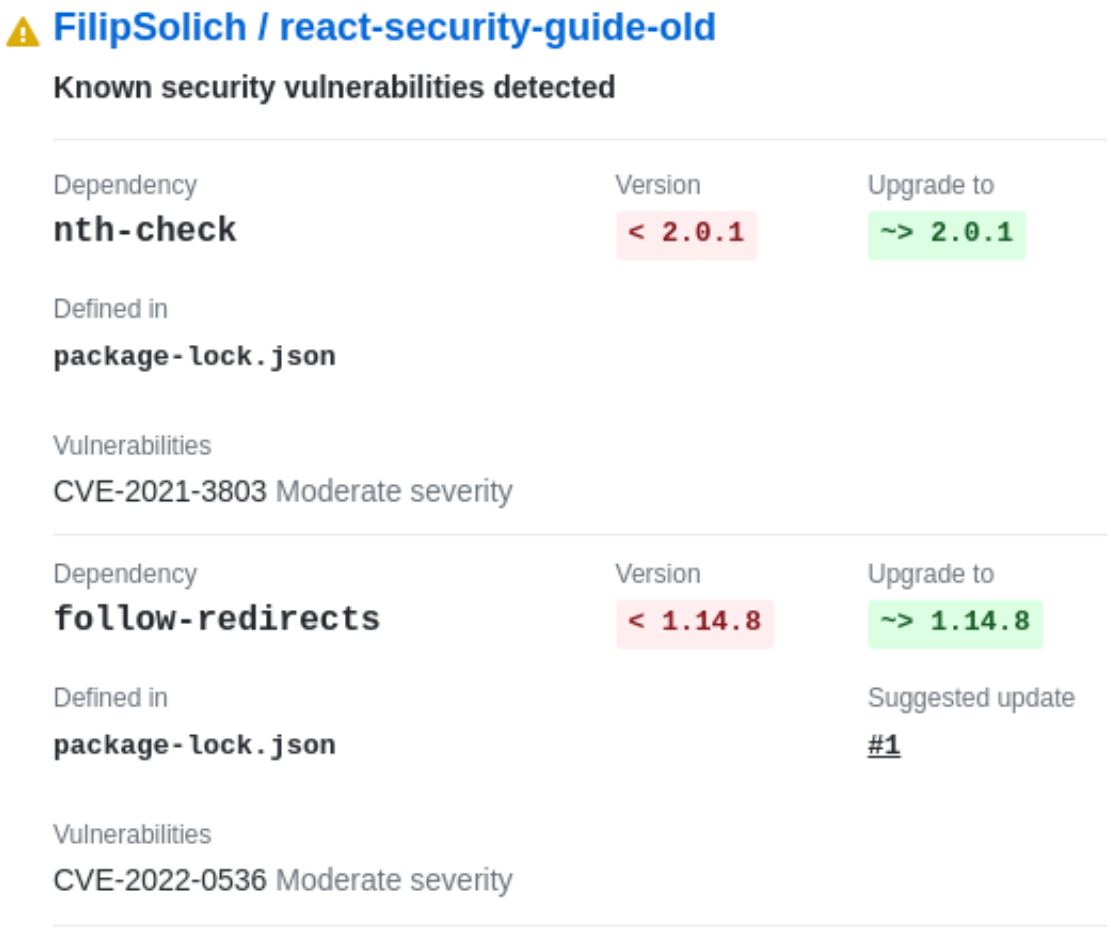
4.3 GitGuardian

Firma GitGuardian[18] vytváří řešení pro kontrolu úniku citlivých informací, jako jsou hesla nebo API klíče vepsané ve zdrojových kódech. Firma pravidelně skenuje veřejné git repozitáře na GitHubu[1], GitLabu[2], Bitbucketu[3] a dalších službách poskytujících hostování git repozitářů. Jakmile objeví v repozitáři API klíč nebo jinou proměnnou, která by měla být neveřejná, tak pošle majiteli email s upozorněním, že se tam tato proměnná nachází, a že by ji měl odstranit a poté změnit. Firma poskytuje i program pro lokální kontrolu zdrojových kódů, který pomůže programátorovi odhalit citlivé informace ve zdrojovém kódu dřív, než tyto informace nahraje do veřejného repozitáře.

4.4 GitHub - Bezpečnostní kontrola repozitáře

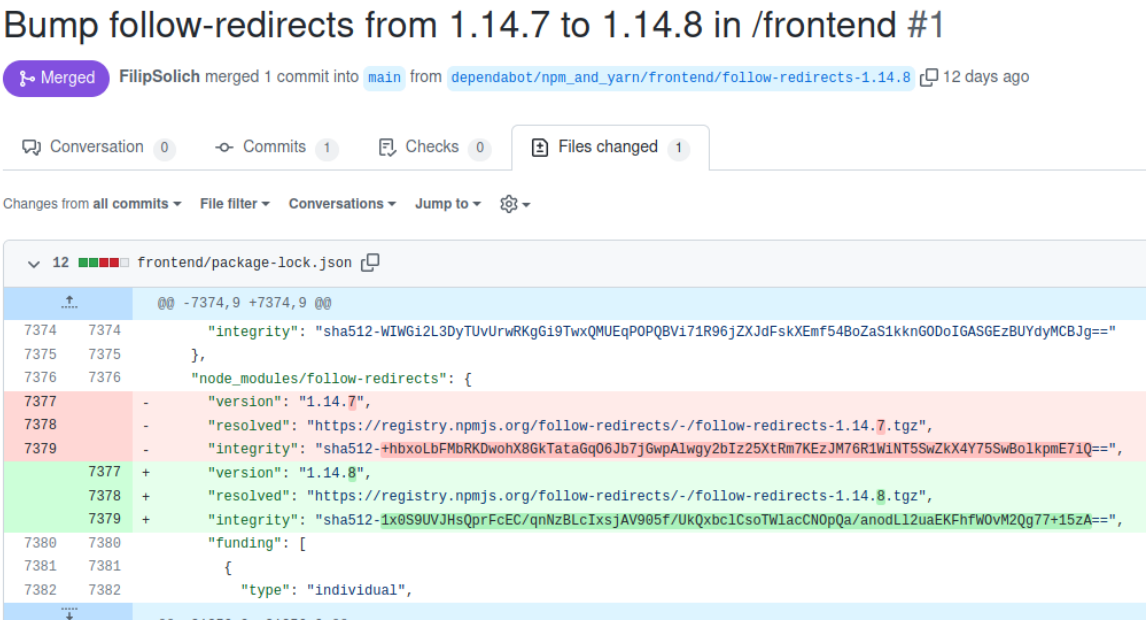
Společnost GitHub poskytuje službu pro správu vzdáleného Git repozitáře, který slouží k verzování zdrojových souborů. GitHub se používá pro vývoj spousty veřejných projektů s otevřeným zdrojovým kódem. To, že je zdrojový kód přístupný každému, znamená, že jej muže jednoduše prozkoumat i útočník a najít v kódu zranitelnost, které může využít. I v případě, že je zdrojový kód na GitHubu soukromý a tedy nemůže ho vidět nikdo kromě vývojářů, kteří na něm pracují, může se stát, že vývojář do repozitáře pošle firemní tajemství v podobě tajného klíče, který by byl viditelný v aplikaci pro koncové uživatele (například by mohl být součástí HTML kódu). Společnost GitHub si tento problém uvědomuje a snaží se vývojářům co nejvíce pomoci. Jeden ze způsobů, jak může GitHub pomoci při vývoji softwaru, je sken závislostí programu. Tento sken je potřeba povolit v nastavení každého repozitáře pomocí nastavení `Settings -> Code security and analysis -> Dependabot alerts`. Když je nastavení aktivní, bude bot `dependabot` pravidelně skenovat soubory `package.json` a `package-lock.json`, ve kterých se nachází seznam závislostí (a jejich verzí) aplikace napsané v jazyku JavaScript. Pokud si bot všimne, že některá závislost v dané verzi má známou zranitelnost v seznamu CVE (Common Vulnerabilities and Exposures - Společné zranitelnosti a vystavení nebezpečí)[2] a zároveň, že daná závislost má vydanou novou verzi s opravou této zranitelnosti, tak na to autora repozitáře upozorní.

[1] https://github.com
[2] https://about.gitlab.com
[3] https://bitbucket.org



Obrázek 4.1: Email zaslaný GitHubem poté, co byly nalezeny zranitelné závislosti v repozitáři

V emailu jsou vidět všechny zranitelné verze závislostí, které lze aktualizovat na novější. Červeně je vyznačena aktuální verze knihovny. V tomto případě je to pouze informace, že verze je nižší než verze, na kterou bychom knihovnu měli aktualizovat. Zeleně je vyznačená verze obsahující opravu zranitelnosti. Součástí reportu je také ID zranitelnosti a její závažnost. U knihovny `nth-check` je ID CVE-2021-3803, kde 2021 je rok, kdy byla zranitelnost nalezena, a 3803 značí pořadí zranitelnosti v roce 2021. Závažnost této zranitelnosti je Moderate severity, to znamená střední závažnost. U knihovny `follow-redirects` je vidět že dependabot navrhnul aktualizaci na novou verzi a vytvořil pro to Pull request (žádost o zahrnutí nové změny do repozitáře) s číslem 1.



Obrázek 4.2: Pull request vytvořen dependabotem

Bot navrhuje změnu verze knihovny v souboru `package-lock.json`. Díky této změně v souboru si bude moct jednoduše každý vývojář pracující na projektu aktualizovat knihovnu `follow-redirects` na novou verzi. Aktualizace knihoven se poté provádí příkazem `npm install ...` nebo `yarn add ...`.

Kapitola 5

Návrh příručky

Součástí této práce je i vytvoření příručky pro programátory vytvářející aplikace v knihovně React. Tato příručka by měla sloužit pro vzdělání programátora v oblasti bezpečnosti webových aplikací a konkrétně aplikací napsaných v knihovně React. Aby byla příručka co nejužitečnější a dostalo se k ní co nejvíce lidí, je dobrou volbou veřejná webová aplikace.

Příručka by kromě informací, které se týkají bezpečného programování, měla obsahovat také nějaký způsob kontroly, kde si uživatelé příručky ověří, zda všemu rozumí. Pro tuto příručku se zdá nejlepší způsob kontroly kvíz, který bude obsahovat otázky k vysvětlenému tématu a několik možností, jak na otázky odpovědět. Po zvolení odpovědi ke každé otázce a jejich odeslání, je server vyhodnotí a ukáže správné odpovědi.

Je možné, že si programátor bude chtít přečíst části příručky a udělat kvíz pro tyto části postupně, a ne najednou. Pro tuto situaci by programátor určitě ocenil možnost zapamatování si svého postup příručkou i zapamatování svých odpovědí v kvízu. Proto bude možnost mít v aplikaci uživatelský účet, se kterým budou asociovány odpovědi uživatele. Jelikož se jedná o malou a jednoduchou aplikaci, je zbytečné, aby se uživatel ke svému účtu přihlašoval tradičním loginem a heslem. Tento způsob by vyžadoval implementaci autentizace a pro jeho využití by si uživatel musel vytvářet nový samostatný účet. Lepším způsobem pro tento typ aplikace je využití autentizace nějakou existující autoritou. Vzhledem k tomu, že je aplikace určena programátorům, bylo by dobré jako autentizační autoritu použít službu, u které již většina programátorů účet má. Ideálními kandidáty jsou služby GitHub a GitLab. Obě tyto služby poskytují vzdálené hostování git repozitářů, které slouží pro verzování zdrojových souborů.

5.1 Architektura aplikace

Aplikace je rozdělena do dvou částí:

- Core - Obsahující samotné články, kvízy a hlavní HTML šablony
- OAuth2 - Obsahující uživatelský model a pohledy pro přihlášení uživatelů pomocí autentizace u služeb třetí strany

5.1.1 `Core`

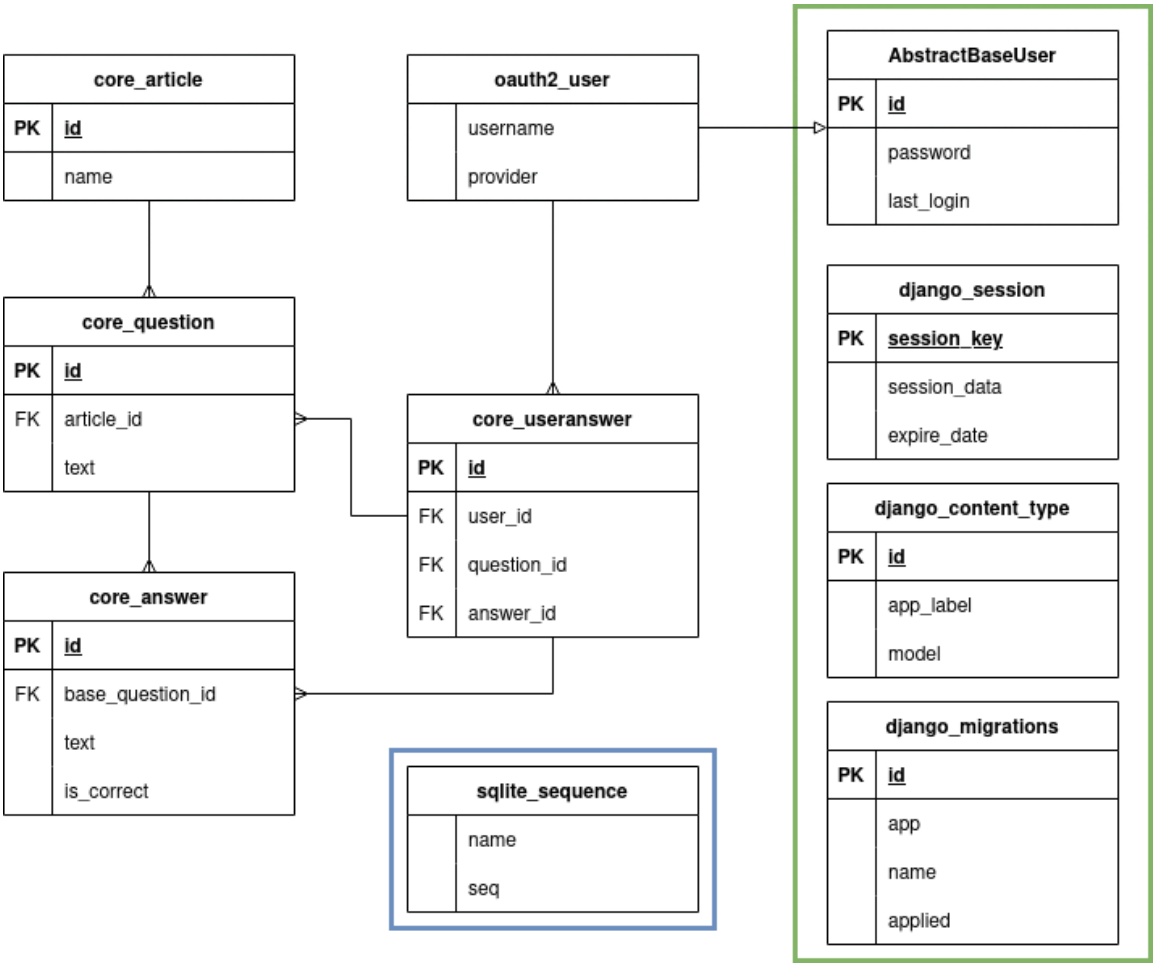
Tato webová aplikace je převážně statická, takže aplikace `Core` obsahuje obecné HTML šablony a jednotlivé články. Dále jsou v aplikaci definované modely pro uchování článků a

kvízů k nim vytvořeným. Z pohledu JavaScritu tento modul obsahuje skript pro zasílání odpovědí na kvíz a následné zobrazení správných a špatných výsledků.

5.1.2 `OAuth2`

Aplikace `OAuth2` je zaměřená na autentizaci uživatelů. Pro využívání aplikace není autentizace potřeba, ale je zde z toho důvodu, aby si uživatel mohl uložit postup v jednotlivých kvízech, které bude aplikace obsahovat. Poté, co se uživatel přihlásí ke svému účtu u služby GitHub nebo GitLab, bude vyzván, aby autorizoval tuto aplikaci (tedy sdělí, že tato aplikace má právo přistupovat k datům daného uživatele).

5.2 Návrh databáze



Obrázek 5.1: Návrh databáze

Databáze ze schématu je vizuálně rozdělena na 3 části. V zeleném obdélníku se nachází tabulky, které vytváří Django (Django vytváří tabulek více, ale nejsou v tomto projektu využity). V modrém obdélníku je tabulka vytvořená databází SQLite. Ostatní tabulky jsou definovány mnou v modelech.

- `core_article` - Tabulka obsahující název článku. Slouží pro vyhledávání existujících článků a shlukování otázek souvisejících s článkem.
- `core_question` - Tabulka obsahující text otázky. Tento text může být formátován pomocí HTML.
- `core_answer` - Tabulka obsahující text odpovědi a informaci, zda se jedná o správnou odpověď k příslušné otázce. Stejně jako u otázky, i text odpovědi lze formátovat pomocí HTML.
- `core_useranswer` - Tabulka obsahující informace o tom, jak uživatel odpověděl na otázku.
- `oauth2_user` - Tabulka reprezentující uživatele. Dědí atributy z Django abstraktního modelu `AbstractBaseUser`. Tabulka obsahuje uživatelské jméno, které jí poskytuje GitHub nebo GitLab.
- `django_session` - Django tabulka uchovávající aktivní sezení uživatelů.
- `django_content_type` - Tabulka obsahující informace o existujících Django aplikacích a jejich modelech.
- `django_migrations` - Tabulka obsahující informace o provedených migracích na databázi. Díky informacím v této tabulce se neprovede stejná migrace na jedné databázi více než jednou.
- `sqlite_sequence` - Tuto tabulku využívá pouze SQLite a obsahuje hodnoty pro automatickou inkrementaci primárních klíčů tabulek.

Kapitola 6

Implementace příručky

Tato kapitola tedy popisuje návrh a implementaci příručky pro vývojáře programující v JavaSriptu a využívající knihovnu React. Je zde popsán výběr technologií pro tvorbu webových aplikací.

6.1 Výběr technologií

V této části jsou popsány využité technologie pro backend a frontend webové aplikace, databáze a využití existujících nástrojů.

6.1.1 Backend - Python, Django

Backend je napsán v programovacím jazyce Python s využitím webového frameworku Django[3]. Django je webový framework zaměřený na rychlost vývoje webových aplikací. Django od základu implementuje spoustu funkcionalit, které jsou potřeba u spousty webových aplikací, a tím ulehčí programátorovi práci. Mezi nejvýznamnější části aplikace poskytnuté frameworkem Django patří databázový model reprezentující uživatele aplikace spolu s autentizací, správou sezení a bezpečnou změnou hesla uživatelem. Dále je třeba zmínit bezpečnost frameworku, který brání před napadením stylu SQL injekce a XSS.

Aplikace v Djangu se strukturují do 3 částí.

- Model - Stará se komunikaci aplikace s databází.
- View - Pohledy zpracovávají přicházející HTTP požadavky a vrací HTTP odpovědi. Obsahuje logiku aplikace.
- Template - Vytváření vzhledu aplikace pomocí HTML šablon.

Zkráceně se tomuto návrhu říká MVT. Samotná aplikace se dále skládá z podaplikací. Každá podaplikace má vlastní modely, pohledy a šablony. Tento způsob umožňuje vzít jednu Django podaplikaci a využít jí v jiném projektu.

Model

Databázové tabulky a vztahy vytváří Django pomocí techniky objektově relačního mapování (ORM - Object-relational mapping). Tato technika umožnuje programátorovi definovat tabulky databáze jako třídy v daném programovacím jazyku, v tomto případě Python třídy.

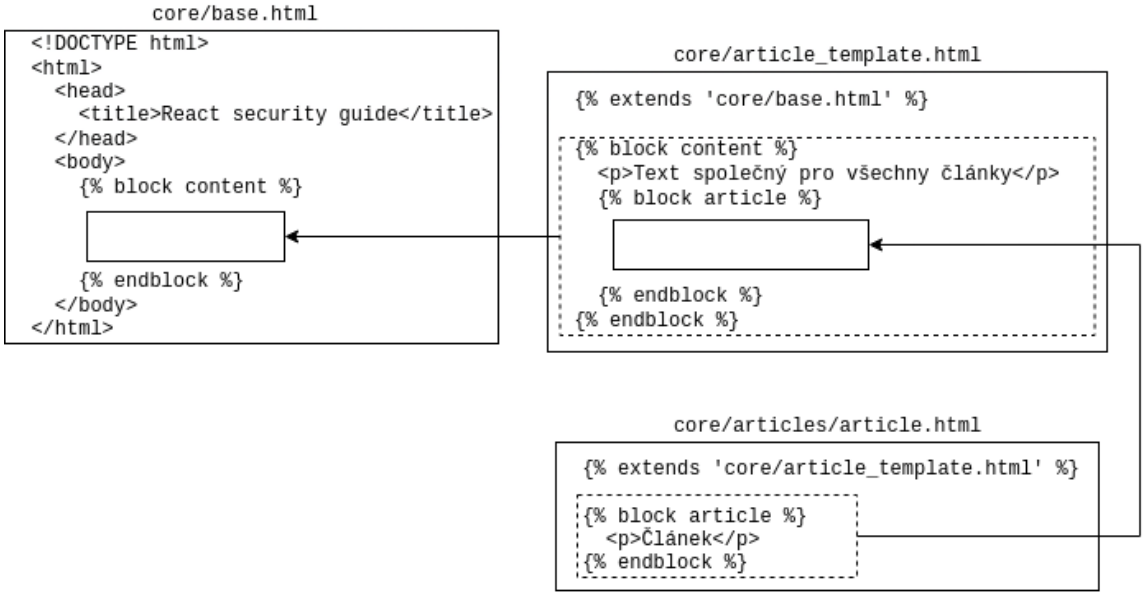
Django poté z programové definice třídy vytvoří odpovídající tabulku v databázi. Díky tomuto způsobu vytváření tabulek se programátor nemusí učit jazyk pro tvorbu databází jako například SQL. Zároveň může s jednotlivými záznamy v tabulce pracovat jako s instancemi třídy a definovat nad těmito třídami metody.

View

Pohledy fungují velice přímočaře. Funkce nebo metoda třídy, která pohled implementuje, dostane HTTP požadavek na vstup v podobě třídy `HttpRequest` a vrací HTTP odpověď v podobě třídy `HttpResponse`. Tělo pohledu tedy implementuje funkcionalitu pro daný HTTP požadavek. Pohledy je třeba registrovat v seznamu povolených URL cest.

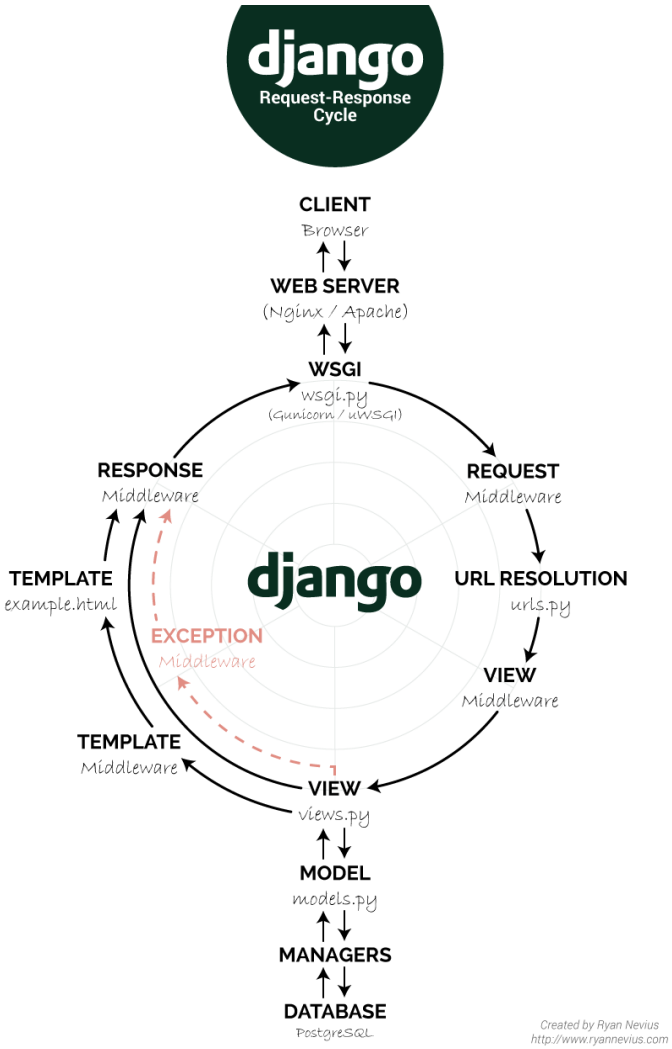
Template

HTML šablony se používají pro dynamické skládání HTML. Šablonovací motor knihovny Django podporuje používání proměnných a základní programové konstrukce jako jsou podmínky a cykly. Spojovat šablony lze bud pomocí vložení jedné šablony do druhé pomocí speciální sekvence `{% include 'app/navbar.html' %}`, nebo lze jednou šablonou doplnit bloky HTML kódu do druhé pomocí sekvence `{% extendes 'app/base.html' %}`.



Obrázek 6.1: Ukázka šablonování HTML v Djangu

Postup HTTP požadavku a odpovědi v Django frameworku



Obrázek 6.2: Cesta HTTP dotazu a odpovědi.[4]

Na obrázku je vidět cesta HTTP požadavku a HTTP odpovědi ve frameworku Django. HTTP požadavek dorazí a je webserverem předán modulu `wsgi.py`. WSGI (Web Server Gateway Interface) je rozhraní mezi web serverem a Python aplikací, které je definováno v PEP 333[7] a PEP 3333[8] pro Python3 aplikace. Modul WSGI vytvoří z příchozího HTTP požadavku instanci třídy `HttpRequest`. Tato instance je dále předána pohledu odpovídajícímu URL cestě. Pohled může být funkce nebo třída, které se na vstup předá `HttpRequest` a ona vrací `HttpResponse`, ze které Django vytvoří odpovídající HTTP odpověď, která je poslána zpět klientovi.

6.1.2 Frontend - HTML, Bootstrap css, jQuery JavaScript

HTML

Pro vytvoření textu a jeho struktury v aplikaci je použit značkovací jazyk HTML (Hypertext Markup Language). Jednotlivé HTML soubory jsou v aplikaci skládány z HTML šablon.

CSS - Bootstrap

CSS je zkratka pro "Cascading Style Sheet". Tento výraz se do češtiny překládá jako "kaskádové styly". Pro kaskádové styly je využit toolkit Bootstrap[1]. Tento toolkit je jeden z nejpoužívanějších pro snadnou tvorbu konzistentních stylů na webových stránkách. Pro jeho zakomponování stačí do HTML vložit odkaz s adresou, odkud se kaskádové styly stáhnou do prohlížeče uživatele. Samotné stylování se poté provádí pomocí HTML atributu `class`, pomocí kterého se nastaví jednotlivým prvkům stránky třídy, kterým Bootstrap definuje vzhled.

JavaScript - jQuery

jQuery[6] je relativně malá JavaScriptová knihovna, která má za cíl usnadnit práci s čistým JavaScriptem. Knihovna je velice vhodná pro nacházení HTML elementů na stránce pomocí selektorů pro nalezení prvků podle jejich `id`, `class` nebo názvu elementu.

Knihovna také ulehčuje zasílání asynchronních HTTP požadavků na webový server a zpracování jejich odpovědí. Technice zasílání asynchronních HTTP požadavků na webserver se říká AJAX (Asynchronous JavaScript And XML). Pomocí AJAXu a JavaScriptu lze měnit obsah stránky bez nutnosti ji znovu celou načíst.

6.1.3 Databáze - SQLite

SQLite[11] je odlehčená databáze, která nepotřebuje separátní serverový proces pro svůj běh. Díky tomu, že databáze ukládá data do jediného souboru a je velice jednoduchá na konfiguraci, je ideální na použití pro jednoduchou aplikaci jako je příručka pro bezpečné programování v Reactu.

6.1.4 Pygments

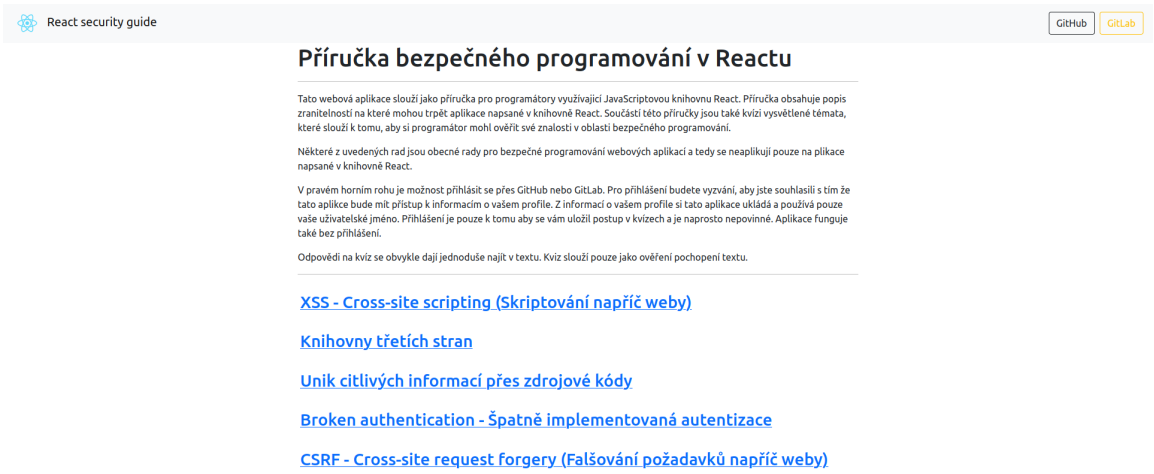
Pygments[9] je knihovna napsaná v jazyce Python a slouží pro generování úryvků kódu se zvýrazněním syntaxe. Program generuje výstup do HTML takže lze úryvky použít pro webové stránky.

6.2 Tvorba aplikace

Tato část popisuje tvorbu aplikace a způsob autentizace uživatelů.

6.2.1 Tvorba příručky a kvízů

Při implementaci příručky, jejíž účel je být přehledným a rychlým zdrojem informací, jsem dbal na to, aby byl design aplikace jednoduchý a přehledný. Titulní strana aplikace tedy neobsahuje nic, co obsahovat nemá. V horní liště aplikace se nachází pouze název aplikace a 2 tlačítka pro přihlášení se, která se změní na login uživatele a tlačítko pro odhlášení, pokud je uživatel již přihlášen. Text celé titulní strany je poté pouze ve středu obrazovky. Na začátku je vysvětlení pro uživatele, co na této stránce najde a jak může příručku používat, a poté následují už jen odkazy na jednotlivé články.



Obrázek 6.3: Hlavní obrazovka příručky

6.2.2 Články v příručce

Články v příručce jsou převzaty z kapitoly "Pokyny pro bezpečné programování"[3] a upraveny tak, aby se dobře četly z pozice uživatele příručky.

Obrázek 6.4: Ukázka článku v příručce

6.2.3 Kvízy

Kvízy jsou navrženy a vytvořeny tak, aby pro jeden článek existoval jeden kvíz, který může obsahovat neomezeně otázek. U každé otázky jsou na výběr tři odpovědi a vždy jen jedna je správná.

Otázky i odpovědi jsou načteny z databáze a je možné je psát v HTML. Díky tomu je možné text v otázce či odpovědi různě stylovat a vytvořit v nich bloky programového kódu.

Pro vytvoření programových bloků kódu s barevným zvýrazněním syntaxe byl použit nástroj `pygments`[9]. Nástroj se dá použít jako knihovna pro jazyk Python nebo jako spustitelná aplikace. Výstup z programu jsem zvolil ve formátu HTML. Příkaz pro vygenerování vypadal následovně.

```
pygmentize -f html -o output.html input.js
```

`-f html` značí, že chceme výstup ve formátu HTML, `output.html` je název souboru, který bude obsahovat výstup a `input.js` je vstupní soubor se zdrojovým kódem, u kterého dokáže program detekovat, v jakém programovacím jazyce je tento vstupní kód napsán.

Odeslání zvolených odpovědí se provádí na pozadí přes AJAX. Server tyto odpovědi vyhodnotí a v případě, že na otázky odpovídal přihlášený uživatel, je server taky uloží. V odpovědi server pošle 2 pole obsahující ID odpovědí. První pole vrácené pod klíčem `green` obsahuje ID všech správných odpovědí. Druhé pole vrácené pod klíčem `red` obsahuje ID odpovědí, které zvolil uživatel špatně. JavaScriptový kód poté odpovědi obarví do barvy podle toho, v jakém poli se ID odpovědi nachází.

6.2.4 Autentizace uživatelů

Aby aplikace mohla ukládat odpovědi k otázkám u konkrétního uživatele, je potřeba každou odpověď s uživatelem asociovat. Pro uživatele, který si bude chtít ukládat své odpovědi, tedy bude vytvořen účet a jeho odpovědi budou asociovány s tímto účtem. Aby si uživatel nemusel účet vytvářet, má možnost se přihlásit ke službě GitHub nebo GitLab. Po tomto přihlášení bude uživatel službou vyzván, aby odpověděl, jestli souhlasí s tím, že tato aplikace bude mít přístup k jeho uživatelským datům. Pokud uživatel souhlasí, bude mu vytvořen v této aplikaci nový účet na základě dat, ke kterým má teď aplikace přístup. Výhodou takto vytvořeného účtu je, že je to účet bez hesla. Kdyby se neznámému útočníkovi podařilo ukrást databázi uživatelů této aplikace, tak se mu nepovede přihlásit se k účtu žádného uživatele.

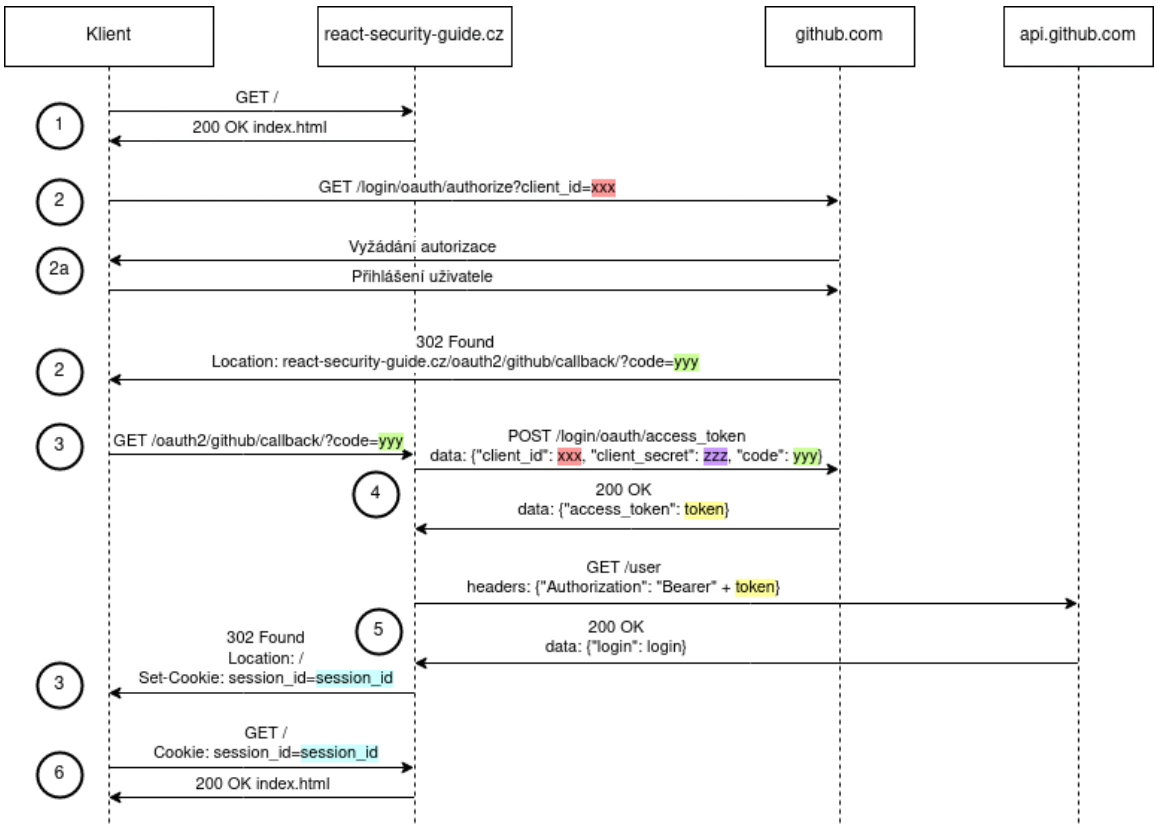
Pokud uživatel tuto příručku v budoucnu navštíví znovu a opět zvolí přihlášení přes stejnou službu, tak už mu nebude vytvořen nový účet, ale bude přihlášen ke svému existujícímu účtu.

Minimální požadavky na uživatelské účty

Uživatelské účty v naší aplikaci nemusí mít o uživateli téměř žádné informace. Jediná informace, kterou tato aplikace potřebuje, je nějaký unikátní identifikátor, pomocí kterého bude možné jednotlivé účty od sebe odlišit. Služba GitHub i GitLab vyžadují po uživateli, aby si při vytváření účtu vymyslel jedinečné jméno, pod kterým na dané službě ještě neexistuje žádný účet. Toto uživatelské jméno tedy můžeme využít jako identifikátor účtu i pro naši aplikaci. Zde ale vzniká jeden problém. Na službě GitHub může třeba existovat účet s jedním uživatelským jménem, ale ve službě GitLab může existovat účet se stejným jménem, ale může patřit jiné osobě. Proto pro účet v naší aplikaci potřebuje další informaci o uživateli, aby bylo možné tyto dvě osoby od sebe odlišit. Dobrým kandidátem pro tento rozšiřující údaj je právě název služby, u které se uživatel autentizoval. Takto vznikne dvojce služba + login, která bude určitě unikátní.

Ukázka autentizace uživatelů přes službu GitHub

Na následujícím obrázku je vyobrazen postup pro autentizaci uživatele přes službu GitHub a komunikace aplikace se servery GitHubu pomocí protokolu OAuth2.



Obrázek 6.5: Autentizace přes službu GitHub

1. Uživatel přistoupí na hlavní stránku této aplikace. Server pošle zpátky HTML soubor úvodní stránky s tlačítky pro přihlášení přes GitHub nebo GitLab

2. Po kliknutí na tlačítko pro autentizaci přes GitHub pošle webový prohlížeč `GET` požadavek autentizační URL služby GitHub. URL obsahuje parametr `client_id`, který byl v HTML předvyplněný backendem aplikace. Pokud je uživatel u služby GitHub přihlášen (Prohlížeč má uložené aktivní sezení pro doménu `github.com`) tak server GitHub vrátí odpověď obsahující HTTP hlavičku `Location`, která obsahuje adresu, na kterou se má klient přesměrovat. Tato adresa vede na backend aplikace a její součástí je parametr `code`. Tento `code` do URL předvyplnil server GitHub a je to unikátní kód pro přihlášeného uživatele signalizující, že se uživatel úspěšně autentizoval, a o jakého uživatele se jedná.

   (a) Pokud uživatel u služby GitHub ještě není přihlášen (prohlížeč nemá aktivní sezení pro doménu `github.com`), vyzve nejdříve server GitHub uživatele k přihlášení se do služby. Poté pokračuje s odpovědí jako u bodu 2.

3. Klient pošle HTTP požadavek na adresu, kterou dostal v odpovědi č. 2 spolu s kódem `code`. Backend poté pomocí dotazů č. 4 a 5 zjistí uživatelské jméno uživatele, a pokud

se uživatel do příručky přihlašuje poprvé, je mu vytvořen účet obsahující uživatelské jméno a název služby, u které se uživatel autentizoval. Odpověď na tento dotaz nastaví sezení pro uživatele a přesměruje jej na hlavní stránku aplikace.

4. Backend aplikace pošle HTTP dotaz s žádostí o přístupový token na server GitHubu. Tento dotaz obsahuje `client_id` (ID této aplikace, které vystavil GitHub), `client_secret` (tajný kód této aplikace, který vystvail GitHub a zná jej pouze backend aplikace, takže se nikdy nedostane jako součást HTML ke klientovi) a `code` (kód, který GitHub vystavil uživateli poté, co se úspěšně autentizoval). GitHub si podle poskytnutého kódu `code` vyhledá, že se skutečně jedná o kód, který vystavil uživateli, který o kód požádal se stejným `client_id`. Poté ověří zda `client_secret` je stejné jako to, které vystavil pro dané `client_id`. Pokud vše sedí, GitHub pozná, že tento požadavek pochází z legitimního serveru, a v HTTP odpovědi vrátí přístupový token `token`, který slouží aplikaci pro přístup k datům přihlášeného uživatele na GitHubu přes GitHub API.

5. Aplikace pošle HTTP dotaz na GitHub API na koncový bod sloužící k získání informací o uživateli. Součástí dotazu je přístupový token `token`. GitHub API pošle HTTP odpověď obsahující uživatelská data ve formátu JSON.

6. Uživatel již má aktivní sezení a může aplikaci používat jako přihlášený uživatel. Sezení se posílá spolu s každým HTTP dotazem na aplikaci.

6.3 Nasazení aplikace

Aby byla aplikace volně přístupná komukoli je potřeba ji spustit na serveru. Starat se o server, tak aby byl přístupný 24 hodin denně, byl zálohovaný a zabezpečený proti útokům je velice náročné a proto hodně jednotlivců i firem využívá externí poskytovatelé, kteří se starají o servery za ně. Já jsem tedy využil služeb poskytovatele Heroku[1]. Heroku jsem zvolil protože poskytuje své služby pro malé neziskové projekty zdarma. Platforma Heroku také poskytuje PostgreSQL[2] databázi zdarma pro studenty a osobní projekty.

Aplikaci na Heroku lze nasadit několika způsoby:

- Nahrání zdrojových kódů přes Git.
- Připojení GitHub účtu odkud si Heroku zdrojové kódy stáhne.
- Nasazení přes sestavený Docker image.

Jako způsob nasazení jsem zvolil nahrávání zdrojových kódu přes vzdálený Git repozitář. Díky tomuto způsobu stačí po každe změně ve zdrojových souborech zadat následující příkazy:

```
git commit -am 'Nova zmena' && git push heroku main
```

Tyto příkazy pošlou změny na server a Heroku automaticky provede akce, které jsou potřeba pro to aby se aplikace spustila.

[1] heroku.com
[2] https://www.postgresql.org/

Nakonec stačí importovat otázky a odpovědi do databáze. Otázky a odpovědi jsou uloženy v souboru typu JSON a jejich import se provede následujícím příkazem

```
python manage.py loaddata db.json
```

Tato aplikace je přístupná na adrese https://react-security-guide.herokuapp.com/

Kapitola 7

Implementace exploitu

Tato kapitola se zabývá návrhem a implementací exploitu využívajícího zranitelnosti v knihovně React. Je zde popsáno o jakou zranitelnost se jedná, jak zranitelnosti využít a jak se jí vyhnout. Dále je zde popsán návrh vhodného prostředí pro vytvoření zranitelné aplikace a příklad útoku na zranitelnou aplikaci.

7.1 XSS na straně serveru

Tato zranitelnost se týká možnosti skriptování mezi weby u některých aplikací využívajících vytváření HTML stránky na serveru místo tradiční metody, kdy se stránka sestaví na klientské straně. Informace o zranitelnosti jsou převzaty z oficialního React blogu[14].

7.1.1 Popis zranitelnosti

Zranitelnost má označení CVE-2018-6341 a postihuje modul `react-dom/server`. Toto CVE dostalo hodnotu nebezpečí 6.1/10[13]. Dobrou zprávou je, že zranitelnost nepostihuje všechny aplikace využívající knihovnu `react-dom`, ale pouze aplikace ve specifickém případě, který je popsán dále. Modul `react-dom/server` slouží pro vytvoření HTML na serveru namísto vytvoření HTML na klientovi. Tato metoda není příliš tradiční, protože vyžaduje aplikaci napsanou v jiné JavaScriptové knihovně, která umožňuje psát backend webových aplikací, jako například knihovna Node.js. Aby aplikace napsaná v Node.js mohla vytvořit HTML z aplikace napsané v Reactu, může využít modul `react-dom/server`, který je vyvíjen společně s Reactem a je součástí balíčku `react-dom`. Tento modul obsahuje metodu `ReactDOMServer.renderToString()`, která vytvoří HTML v řetězci z poskytnuté React komponenty v argumentu. Příklad použití: `ReactDOMServer.renderToString(<App />)` vytvoří HTML z komponenty `App` přímo na serveru.

Právě metoda `renderToString` špatně ošetřuje část dat v elemetu, který je předán jako argument funkce. Princip fungování metody je ukázán na následujícím příkladu.

```
1 import ReactDOMServer from 'react-dom/server'
2
3 let props = {'id': 'my-id'}
4 let element = <div {...props} />
5 let html = ReactDOMServer.renderToString(element)
6
7 // `html` obsahuje "<div id="my-id" data-reactroot=""></div>"
```

Na řádku 3 se vytváří JavaScriptový objekt `props`, který má pod klíčem `id` uloženou hodnotu `my-id`. Na řádku 4 se pomocí JSX syntaxe vytvoří `div`, u kterého je specifiko-

váno, že jako atributy elementu budou použity klíče a hodnoty objektu `props`. Na řádku číslo 5 se voláním `ReactDOMServer.renderToString(element)` vytvoří řetězec obsahující vytvořený `div`. Tento `div` obsahuje atributy vytvořené z objektu `props` a dále atribut `data-reactroot=""`, který doplnil sám React a je pro tuto ukázku nepodstatný.

Tento kód sám o sobě problematický není. Problém vzniká, jakmile klíč objektu `props` je zadán uživatelem, tak jak je ukázáno na následujícím příkladu.

```
1  import ReactDOMServer from 'react-dom/server'
2
3  let userProvidedData = '></div><script>alert("hi")</script>'
4
5  let props = {}
6  props[userProvidedData] = 'my-id'
7  let element = <div {...props} />
8  let html = ReactDOMServer.renderToString(element)
9
10 // `html` "obsahuje <div ></div><script>alert('hi')</script>="my-id" data-reactroot="">"
```

Předpokládejme, že data v proměnné `userProvidedData` na řádku 3 jsou data, která zadal uživatel aplikace třeba přes formulář. Takto zadaná data by vedla k tomu, že se na řádku 6 do proměnné `props` přidá položka s klíčem, který odpovídá těmto datům, a s hodnotou, které bude stále `my-id`. `userProvidedData` tedy bude obsahovat:
`{'></div><script>alert("hi")</script>': 'my-id'}`

Jakmile se tento objekt použije pro vyplnění atributu elementu, který bude následně přeložen do HTML řetězce, vznikne HTML element obsahující JavaScriptový kód. Text `></div>` na začátku klíče uzavře `div`, do kterého jsou data doplňována a zbytek klíče obsahující `<script>alert("hi")</script>` bude přidán jako element za `div`. Tento skript může obsahovat libovolný JavaScriptový kód.

7.1.2 Oprava zranitelnosti

Opravit tuto zranitelnost ve zdrojových kódech Reactu je triviální a již takto opravena byla.

Obrázek 7.1: Oprava v commitu 5b19684fc3eddb44a790f31804707de9234147c7[15]

V přidané podmínce se kontroluje klíč, který bude použit jako název atributu tím, že se zavolá funkce `isAttributeNameSafe()` s názvem atributu předaným jako parametr. Pokud

funkce vyhodnotí, že daný název není pro atribut bezpečný, vrátí hodnotu `false` a atribut bude nahrazen prázdným řetězcem.

7.1.3 Zranitelné verze modulu `react-dom`

Seznam zranitelných verzí modulu a jejich opravené alternativy

- 16.0.x
  - 16.0.0 -> 16.0.1
- 16.1.x
  - 16.1.0 -> 16.1.2
  - 16.1.1 -> 16.1.2
- 16.2.x
  - 16.2.0 -> 16.2.1
- 16.3.x
  - 16.3.0 -> 16.3.3
  - 16.3.1 -> 16.3.3
  - 16.3.2 -> 16.3.3
- 16.4.x
  - 16.4.0 -> 16.4.2
  - 16.4.1 -> 16.4.2

Aktualizaci lze provést pomocí `npm` nebo `yarn`

- npm - `npm i react-dom@16.4.2`
- yarn - `yarn add react-dom@16.4.2`

7.1.4 Návrh a použití exploitu

Pro simulaci zneužití zranitelnosti je třeba vytvořit aplikaci obsahující zranitelnost, která by se mohla vyskytnout v opravdové aplikaci. A provést útok na tuto aplikaci. V tomto případě jsou součástí útoku pouze data, která se vloží do formuláře, odesílaného pro zpracování do aplikace.

Tvorba zranitelné aplikace

Aby aplikace byla zranitelná, musí využít balíčku `react-dom` ve verzi, která chybu obsahuje. Pro příklad tedy bude využita verze 16.0.0 a v této verzi bude využit balíček `react`. Dále bude potřeba JavaScriptové knihovny pro tvorbu backendu webových aplikací. Pro tento příklad bude použit framework Express využívající Node.js. Express umožňuje psaní velice jednoduchých aplikaci na pár řádcích kódu, takže je ideální pro prezentaci zranitelnosti. Framework Express je ještě třeba doplnit o balíček `body-parser`, který bude využit pro vytažení dat z těla HTTP požadavku.

`client` Část aplikace ve složce `client` obsahuje kód, který by sám o sobě mohl sloužit jako React aplikace vykreslená v prohlížeči uživatele. Tato React aplikace má jednu React komponentu `App`, která obsahuje HTML se základním popisem této zranitelnosti, a formulář s jedním textovým polem. Textové pole je určeno pro uživatelský vstup, který bude použit jako název atributu `div` elementu.

`server` Část aplikace ve složce `server` je backend aplikace využívající Express v jediném souboru `server.js`. Tato aplikace se spouští namísto klasické React aplikace. Po spuštění bude aplikace dostupná na portu 5000 a přijímá pouze HTTP požadavky vedoucí na URL cestu `"/"`.

Pokud na tuto webovou aplikaci přijde HTTP požadavek s GET metodou, aplikace pomocí ReactDOMServer API vytvoří HTML z React komponenty `App`, která je do skriptu `server.js` importována ze složky `client`. Toto HTML je poté vloženo do souboru `client/index.html`, který obsahuje základní HTML šablonu bez těla. Toto vytvořené HTML je vráceno v HTTP odpovědi.

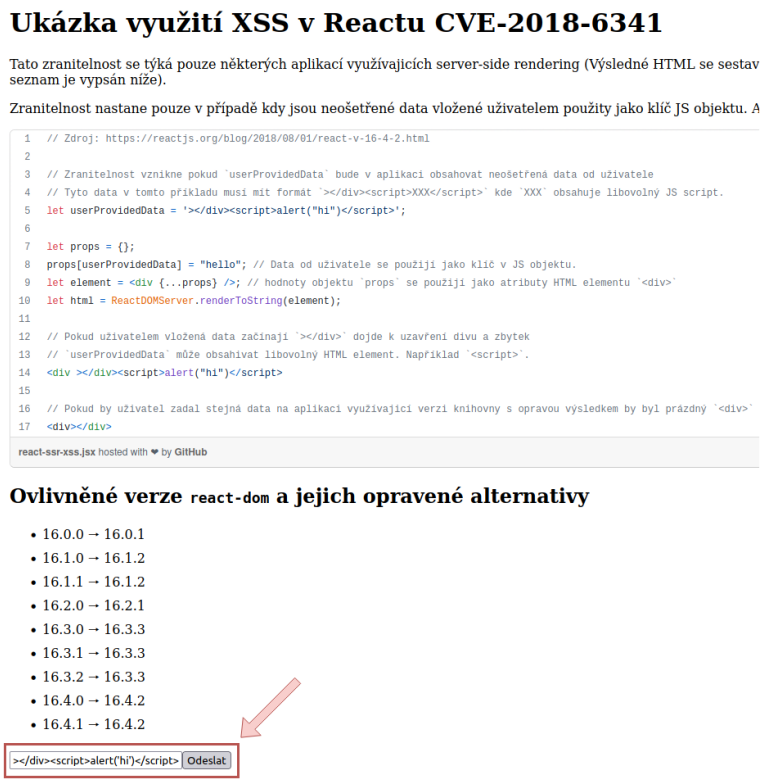
Jakmile uživatel vyplní formulář a odešle jej, server vytvoří stejnou HTML stránku, ale na konec přidá `div` obsahující potenciálně škodlivá data.

```
let maliciousHtml = '' // Promenna neobsahuje zadne data pokud nebyl odeslan formular

if (req.body && req.body.text) {
  let userProvidedData = req.body.text // Precteni dat z formulare
  let props = {}
  props[userProvidedData] = 'hello'
  let element = <div {...props} />
  maliciousHtml = ReactDOMServer.renderToString(element)
}

// Obsah ‘maliciousHtml‘ je pote vlozen do HTML kodu stranky.
```

Tento kód znázorňuje, jak jsou data od uživatele použita pro vytvoření závadného `div` elementu.



Obrázek 7.2: Aplikace demonstrující zranitelnost

Na tomto obrázku je ukázán vzhled aplikace, která demonstruje zneužití zranitelnosti. Formulář je předvyplněn s hodnotou, díky které je do HTML stránky vložen skript, který otevře okno se zprávou "hi".

Kapitola 8

Testování

V této aplikaci je popsán popis testování aplikací, které vznikli jako součást této práce. Testování obsahuje 2 hlavní části. V první části je popsáno testování webové aplikace příručka pro programátory a v druhé části je popsáno testování exploitu.

8.1 Příručka

Na příručce jsem testoval převážně funkčnost autentizace a vyhodnocování a ukládání výsledků kvízu.

Jako nepřihlášený uživatel jsem odpověděl na otázky v kvízu a kviz odeslal. Správně odpovědi se zbarvily do zelena podle očekávání a špatně zodpovězené otázky se zbarvili do červena. Po znovu načtení stránky se odpovědi odbarvili protože se výsledky pro nepřihlášeného uživatele nikde neukládají.

Tento scénář jsem opakoval i jako přihlášený uživatel a znovu načtení stránky zůstali odpovědi barevně označené podle správnosti.

8.2 Exploit

Tato sekce obsahuje popis testování exploitu. Demonstrační aplikace je naprosto oddělená od příručky a proto jsou také obě aplikace testovány separátně.

8.2.1 Spusťění explotu

Pro testování exploitu bylo třeba spustit aplikaci obsahující exploit lokálně na mém počítači. Aby bylo možné exlpoit spustit lokálně je třeba nainstalovat všechny závislosti, které jsou definované v souboru `package.json`. Instalace závislostí jsem provedl pomocí příkazu `npm install`, který automaticky vyhledá soubor `package.json` a nainstaluje všechny závislosti.

Před spuštěním je třeba JavaSciprtové soubory ve kterých je napsán backend aplikace sestavit do jednoho, který bude spustitelný programem `Node.js`. To jsem provedl příkazem `npm run build`. Tento příkaz vytvořil adresář `dist`, která obsahuje sestavené soubory.

V tuto fázi je vše připraveno a spustil jsem aplikaci z adresáře `dist` pomocí programu `Node.js` příkazem `node server.bundle.js`. Po úspěšné spuštění byla tato aplikace přístupná na adrese `localhost:5000`.

8.2.2 Testování funkčnosti exploitu

Pro otestování funkčnosti jsem otevřel aplikaci, která má ve své spodní části formulář s textovým polem. Toto pole slouží pro vkládání zákeřných řetězců, které způsobí vložení JavaScriptového kódu to stránky. Pro ukázku je formulář již předvyplněn tak aby po se po odeslání formuláře spustil skript, který otevře okno s hláškou. V rámci testování tedy odeslal formulář a skript se úspěšně spustil jak jde vidět na následujícím obrázku.

Obrázek 8.1: Ukázka spuštěného skriptu

Pro ověření že ukázka nefunguje jen pro přednastavenou hodnotu jsem do formuláře napsal následující řetězec:

```
></div><script>let a=5;alert(a)</script>
```

Začátek příkazu musí byt stejný aby se dala zranitelnost zneužít a proto jsem změnil jen obsah elementu `<script>`. Pod odeslání formuláře se opět spustilo okno, které spouští příkaz `alert`, ale tentokrát v něm bylo napsáno `5`

Nakonec jsem otestoval napadení aplikace za použití jiného HTML elementu než je `div`. Tento test vyžadoval změnit zdrojový kód backendové části aplikace.

```
- let element = <div {...props} />
+ let element = <p {...props} />
```

Z kódu jde vidět že řádek začínající znakem `"-"` jsem nahradil za řádek začínající znakem `"+"`. Po této změně jsem opět sestavil aplikaci a spustil. Pro otestování jsem upravil i škodlivý řetězec se skriptem na:

```
></p><script>alert('hi')</script>
```

Po stisknutí tlačítka odeslat se opět otevřelo okno s textem `"hi"`.

Kapitola 9

Závěr

Cílem této práce bylo vytvořit příručku pro programátory využívající JavaScriptovou knihovnu React, a dále vytvořit ukázkový exploit některé ze zranitelností knihovny. Příručku i ukázku exploitu jsem implementoval jako dvě oddělené aplikace. Při psaní práce jsem prostudoval problematiku bezpečného programování v knihovně React, její slabá místa a způsoby zneužití těchto slabých míst. Dále jsem nastudoval doporučenou literaturu a existující nástroje pro bezpečnější programovaní. Jako součást práce jsem navrhnul a implementoval webovou aplikaci, která slouží jako příručka bezpečného programování a jednoduchou webovou aplikaci, která slouží jako ukázka využití exploitu. Při tvorbě práce jsem se naučil programovat frontend webových aplikací za použití knihovny React a nahlížet na vývoj webových aplikací z pohledu bezpečnosti. V práci bych dále pokračoval do hlubšího rozboru skriptování napříč weby, protože tento typ zranitelností postihuje frontendové knihovny nejvíce, a je zde ještě mnoho obsahu, který v této práci pokryt nebyl. Příručka by se v budoucnu dala doplnit o funkční spustitelné příklady jednotlivých zranitelností, které by do aplikace zadal přímo uživatel příručky. Tato funkcionalita by byla velice nápomocná uživateli, protože by si mohl vše, co se naučil, ihned vyzkoušet, a tedy i lépe pochopit. Vzhledem k tomu, že by toto rozšíření dovolovalo v omezené míře "útočit"na aplikaci, bylo by potřeba samotnou příručku zabezpečit na perfektní úrovni tak, aby byl ukázkový útok stále proveditelný, ale nesměl by ohrozit funkčnost příručky nebo ohrozit bezpečí dalších uživatelů příručky.

Literatura

[1] *Bootstrap* [online]. Dostupné z: https://getbootstrap.com/.

[2] *CVE* [online]. Dostupné z: https://cve.mitre.org/cve/.

[3] *Django* [online]. Dostupné z: https://djangoproject.com.

[4] *Django Request-Response Cycle* [online]. Dostupné z: https://www.reddit.com/r/django/comments/2vjyrc/djangos_requestresponse_cycle_a_visual_guide/.

[5] *Gatsby* [online]. Dostupné z: https://github.com/gatsbyjs/gatsby.

[6] *JQuery* [online]. Dostupné z: https://jquery.com/.

[7] *PEP 333 – Python Web Server Gateway Interface v1.0* [online]. Dostupné z: https://peps.python.org/pep-0333/.

[8] *PEP 3333 – Python Web Server Gateway Interface v1.0.1* [online]. Dostupné z: https://peps.python.org/pep-3333/.

[9] *Pygments* [online]. Dostupné z: https://pygments.org.

[10] *Spring* [online]. Dostupné z: https://github.com/pmndrs/react-spring.

[11] *SQLite* [online]. Dostupné z: https://sqlite.org/index.html.

[12] *About NIST*. Gaithersburg: NIST, 2009. Dostupné z: https://www.nist.gov/about-nist.

[13] *CVE-2018-6341* [online]. 2018. Dostupné z: https://www.opencve.io/cve/CVE-2018-6341.

[14] Abramov, D. *React v16.4.2: Server-side vulnerability fix*. 2018. Dostupné z: https://reactjs.org/blog/2018/08/01/react-v-16-4-2.html.

[15] Abramov, D. *Sanitize unknown attribute names for SSR* [online]. 2018. Dostupné z: https://github.com/facebook/react/commit/5b19684fc3eddb44a790f31804707de9234147c7.

[16] Bunge, B. *Adopting Typescript at Scale* [online]. 2019. Dostupné z: https://www.youtube.com/watch?v=P-J9Eg7hJwE.

[17] Düüna, K. *Node-esapi* [online]. 0.0.1. Leden 2014 [cit. 27. 12. 2021]. Dostupné z: https://github.com/ESAPI/node-esapi.

[18] GitGuardian. *GitGuardian* [online]. 2020. Dostupné z: https://www.gitguardian.com/.

[19] Manico, J. *DOM based XSS Prevention Cheat Sheet* [online]. Prosinec 2018 [cit. 27. 12. 2021]. Dostupné z: https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/DOM_based_XSS_Prevention_Cheat_Sheet.md.

[20] Meta. *Introducing JSX* [online]. 2013. Dostupné z: https://reactjs.org/docs/introducing-jsx.html.

[21] Meta. *JSX In Depth* [online]. 2013. Dostupné z: https://reactjs.org/docs/jsx-in-depth.html.

[22] Meta. *React* [online]. 2013. Dostupné z: https://reactjs.org.

[23] Microsoft. *TypeScript* [online]. 2012. Dostupné z: https://www.typescriptlang.org/.

[24] Nonnenberg, S. *Move to react for newlines, emoji, and links in message body* [online]. 2018. Dostupné z: https://github.com/signalapp/Signal-Desktop/commit/4e5c8965ff72576a9e20850dd30d9985f4073192.

[25] OWASP. *OWASP* [online]. 2001. Dostupné z: https://owasp.org/.

[26] OWASP. *OWASP Cheat Sheet Series* [online]. 2021. Dostupné z: https://cheatsheetseries.owasp.org/.

[27] OWASP. *OWASP Top 10:2021* [online]. 2021. Dostupné z: https://owasp.org/Top10/.

[28] Perris, R. *10 React security best practices* [online]. Říjen 2020. Dostupné z: https://snyk.io/blog/10-react-security-best-practices/.

[29] ROSS, R., PILLITTERI, V., GRAUBART, R., BODEAU, D. a MCQUAID, R. Developing Cyber-Resilient Systems: A Systems Security Engineering Approach. *Developing Cyber-Resilient Systems: A Systems Security Engineering Approach*. NIST. 2021, sv. 2021, č. 2, s. 294. DOI: NIST.SP.800-160v2r1. Dostupné z: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-160v2r1.pdf.

[30] Signal, a. c. n. *Signal* [online]. 2013 [cit. 27. 12. 2021]. Dostupné z: https://signal.org/.

Příloha A

Obsah přiloženého paměťového média

- `latex_src/` - zdrojové soubory LaTeX
- `src/` - zdrojové soubory webové aplikace - příručka
- `exploit_src/` - zdrojové soubory exploitu
- `xsolic00.pdf` - text práce ve formátu PDF
- `README.md` - informace o souborech