VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

SLUŽBA PRO ZJIŠTĚNÍ ZRANITELNOSTÍ KNIHOVEN POUŽITÝCH NA WEBOVÝCH STRÁNKÁCH

VULNERABILITY DETECTION SERVICE OF WEB PAGE LIBRARIES

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **RADEK BEDNÁŘ**
AUTHOR

VEDOUCÍ PRÁCE **Ing. TOMÁŠ VOLF**
SUPERVISOR

BRNO 2019

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav informačních systémů (UIFS) Akademický rok 2018/2019

Zadání bakalářské práce

Student: **Bednář Radek**
Program: Informační technologie
Název: **Služba pro zjištění zranitelností knihoven použitých na webových stránkách**
Vulnerability Detection Service of Web Page Libraries
Kategorie: Analýza a testování softwaru

Zadání:

1. Seznamte se s možnostmi detekce použitých komponent / knihoven na webových stránkách.
2. Seznamte se s dostupnými databázemi zranitelností a možnostmi, jak v nich vyhledávat SW zranitelnosti a ty dále zpracovávat.
3. Seznamte se s knihovnami pro zpracování vzdálených stránek a pro práci s DOM stromem.
4. Navrhněte systém (webovou službu), který pro zadanou nebo nahranou stránku umožní zvolit nebo přímo detekovat použité komponenty / knihovny a ukáže pro ně seznam zranitelností. Navrhněte vhodné zpracování seznamů zranitelností a jejich aktualizaci.
5. Navržený systém implementujte, volbu použitých technologií zdůvodněte.
6. Ověřte funkčnost služby na vhodně zvolených webových stránkách.
7. Zhodnoťte dosažené výsledky, diskutujte další možné rozšiřování služby.

Literatura:

- Ward, J.: Instant PHP Web Scraping, Packt Publishing, 2013, ISBN: 9781782164760
- Kosek, J.: PHP a XML, Grada Publishing, 2009, ISBN: 978-80-247-1116-4
- databáze zranitelností: https://snyk.io/vuln/, https://nvd.nist.gov/, https://cwe.mitre.org/index.html, https://www.cvedetails.com/, https://vuldb.com/

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 až 4 zadání. Pro získání zápočtu za SP není vyžadován detailní / finální návrh.

Podrobné závazné pokyny pro vypracování práce viz http://www.fit.vutbr.cz/info/szz/

Vedoucí práce: **Volf Tomáš, Ing.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 15. května 2019
Datum schválení: 28. října 2018

Abstrakt

Tato práce se zabývá vytvořením aplikace pro detekci technologií použitých na webových stránkách a následnému zjištění jejich zranitelností. Aplikace je implementována s využitím Symfony Frameworku a knihovny React.js. Zdrojem informací o zranitelnostech je databáze NVD spolu s daty ze služby GitHub. Kromě detekce technologií, umožňuje aplikace uživateli manuálně vytvářet vlastní sady technologií a ty poté sdílet pomocí URL adresy.

Abstract

This thesis deals with the creating of an application for the detection of technologies used on websites and finding their vulnerabilities. Application is implemented using the Symfony Framework and the React.js library. The information source is the NVD database joined by data from the GitHub service. Apart from the detection of technologies, the application allows the user to manually create his own sets of technologies and share them using the URL address.

Klíčová slova

webová aplikace, zranitelnosti, detekce technologií, CVE, NVD, Symfony Framework, React.js, Zombie.js

Keywords

web application, vulnerabilities, technology detection, CVE, NVD, Symfony Framework, React.js, Zombie.js

Citace

BEDNÁŘ, Radek. *Služba pro zjištění zranitelností knihoven použitých na webových stránkách.* Brno, 2019. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Tomáš Volf

Služba pro zjištění zranitelností knihoven použitých na webových stránkách

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Tomáše Volfa. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Radek Bednář
16. května 2019

Poděkování

Rád bych poděkoval panu Ing. Tomáši Volfovi za velmi cenné rady a věcné připomínky během vedení mé bakalářské práce.

Obsah

Kapitola 1

Úvod

Hlavním cílem mé práce je vytvoření aplikace pro detekci webových technologií a následné vyhledání jejich zranitelností. Tato aplikace může sloužit jak vývojářům, tak i provozovatelům webových stránek. Budou moci rychle a přehledně vidět, jaké zranitelnosti obsahují technologie, které ve svém projektu používají.

Aktuální velké množství *open source* projektů pro vytváření webových stránek umožňuje pohodlně a rychle bez hlubších znalostí sestavit webovou stránku podle vlastních potřeb. Bohužel tento pohodlný způsob upozaďuje nutnost nezapomínat na bezpečnostní rizika, která mohou použitím některých knihoven vzniknout.

Aplikace poběží v prohlížeči a bude tak dostupná na široké škále zařízení. Jediným potřebným vstupem od uživatele bude adresa stránky, u níž bude chtít provést detekci, a následně zjistit zranitelnosti. Kromě specifikace konkrétní stránky, bude možné zadat i konkrétní technologie (případně sadu technologií), což může být velice přínosné při rozhodování jaké technologie na svém projektu použít.

Kapitola 2 popisuje základní webové technologie, existující nástroje pro detekci na základě zadané adresy a také způsoby, jakými lze technologie detekovat. Kapitola 3 přibližuje pojmy jako softwarová slabina a zranitelnost, dává stručný přehled databází zranitelností a také vysvětluje formát, ve kterém jsou zranitelnosti v databázích uchovávány. Kapitola 4 obsahuje informace o postupech a knihovnách, které lze využít pro zpracování webových stránek. Kapitola 5 je věnována návrhu architektury, výběru implementačních technologií a také návrhu schématu aplikační databáze. Kapitola 6 popisuje zajímavé části implementace služeb a uživatelského rozhraní navržené aplikace. Kapitola 7 přináší srovnání vytvořené aplikace s existujícími nástroji pro vyhodnocení úspěšnosti detekce navržené a implementované aplikace.

Kapitola 2

Detekce webových technologií

Tato kapitola popisuje základní webové technologie a možnosti jejich detekce, ať už pomocí existujících nástrojů nebo pomocí principů, které tyto nástroje využívají a budou využity i při implementaci výsledné aplikace, jejíž vytvoření je cílem mé bakalářské práce.

2.1 Základy webových technologií

V této kapitole jsou popsány principy, které jsou základem každé webové stránky nebo aplikace. Jejich pochopení je tedy nutné pro správnou orientaci v dalším obsahu práce.

První část se zaměřuje na protokol HTTP, který je základem každé komunikace mezi internetovým prohlížečem a webovým serverem, další dvě části se poté věnují nejpoužívanějším jazykům pro tvorbu webových stránek - HTML a JavaScriptu.

2.1.1 Protokol HTTP/HTTPS

HTTP je internetový protokol, využívaný při komunikaci mezi klientem a serverem. Každá komunikace pomocí HTTP protokolu se skládá z požadavku a k němu příslušné odpovědi. Obvykle, ne však vždy, probíhá komunikace na portu 80. Protokol využívá TCP, zajišťující spolehlivé doručení.

HTTP požadavek

HTTP požadavek je na začátku každé komunikace mezi klientem a webovým serverem. Hlavní vlastností požadavku je jeho typ, kdy ve verzi HTTP/1.1 jsou k dispozici požadavky typu `GET`, `HEAD`, `POST`, `PUT`, `DELETE`, `OPTIONS`, `TRACE`, `CONNECT`.

Požadavek typu `GET` slouží k získání informace identifikované pomocí URI (Unique Resource Identifier). Tento typ požadavku má omezenou délku zprávy, ukládá se do historie prohlížeče a je možné jej ukládat do cache. Požadavek typu `HEAD` je požadavku `GET` velice podobný, liší se však tím, že odpověď na požadavek typu `HEAD` obsahuje pouze hlavičky a nikoliv tělo. Toto je výhodné pokud potřebujeme zjistit pouze základní informace o daném zdroji, které jsou obsaženy v hlavičkách.

Požadavek typu `POST` je využíván pro úpravu nebo aktualizaci zdrojů. Oproti požadavku typu `GET` nemá tento typ omezenou délku zprávy, a z principu jeho využití nelze požadavky tohoto typu ukládat do cache. Dnešní prohlížeče jsou schopné v určitých situacích uložit odeslaná data a umožnit v případě potřeby opakované odeslání. Typickým využitím požadavku typu `POST` může být odesílání formulářů. Velice podobný požadavku typu `POST`,

je požadavek typu `PUT`. Narozdíl od požadavku `POST` však slouží k vytváření nebo přepisu zdrojů. Aby byl výčet možných operací se zdroji kompletní, ke smazání zdroje lze použít požadavek typu `DELETE`.

Požadavek typu `OPTIONS` slouží k získání informací o možnostech daného zdroje. Můžeme pomocí něj například zjistit jaké typy požadavku daný zdroj podporuje. Výhodou tohoto požadavku je, že umožňuje klientovi zjistit informace o zdroji, aniž by jej musel zdroj přímo získat například pomocí požadavku `GET`.

HTTP odpověd

HTTP odpověď tvoří druhou část komunikace mezi klientem a serverem. Skládá se z hlaviček a poté těla odpovědi. Tělo, jehož délka je určena hlavičkou `Content-Length`, může obsahovat celou webovou stránku, její část nebo datový či binární výstup.

Důležitou vlastností odpovědi je povinný stavový kód odpovědi. Protokol HTTP podporuje kolem čtyřiceti různých stavových kódů, které je možné v odpovědi zaslat. Stavové kódy je velmi důležité posílat ve správném kontextu, protože je na nich závislé, jak bude odpověď klientem zpracována. Podle prvního čísla stavového kódu lze tyto kódy dělit na třídy.

První třídu představují kódy `1xx`, které jsou označovány jako informativní. Odpovědi se stavovým kódem této třídy jsou brány jako dočasné. Další třída je tvořena kódy `2xx`, které reprezentují úspěšné přijetí a následné zpracování požadavku. Třetí třída s kódy `3xx`, slouží k informování klienta o potřebě další akce pro dosažení cíle, jde tedy typicky o přesměrování. Třídu s kódy `4xx`, využívá server v případě, že je chyba na straně klienta. Jde tedy například o situace, kdy požadavek není ve správném formátu nebo klient požaduje neexistující zdroj. Poslední třídou je třída s kódy `5xx`, kterou server oznamuje, že došlo k chybě na jeho straně.

Zabezpečení protokolu HTTP

Důležitým rozšířením protokolu HTTP je jeho zabezpečená varianta HTTPS, která kromě protokolu HTTP využívá i protokol TLS (Transport Layer Security) pro šifrování komunikace. Protokol TLS poskytuje typicky autentizaci serveru pomocí certifikátu. Pokud je certifikát vydán ověřenou certifikační autoritou, klient si může být jistý s kým opravdu komunikuje.

V dnešní době by mělo být použití HTTPS samozřejmostí u každé webové stránky nebo aplikace, už jen z důvodu, že komunikaci díky šifrování není možné odposlouchávat. Zároveň prohlížeče v posledních verzích označují stránky běžící pouze na protokolu HTTP jako nezabezpečené.

Bezstavovost protokolu HTTP

Důležitou vlastností protokolu HTTP je bezstavovost, není tedy schopen u několika požadavků rozpoznat zda pochází od stejného uživatele. K tomuto účelu byla vyvinuta technologie zvaná Cookies.

Díky nim, je možné uložit na straně klienta jednoznačný identifikátor dané relace (často označované jako *sezení/session*) získaný od serveru, a tím tak vytvořit souvislost mezi několika požadavky.

2.1.2 Jazyk HTML

HTML je jazyk pro tvorbu webových stránek. Je pomocí něj možné definovat strukturu webové stránky a její obsah pomocí tzv. značek. V HTML lze skládat výslednou webovou stránku z jednotlivých elementů, kterým lze definovat další atributy. Výsledný HTML kód zpracuje prohlížeč a vytvoří z něj DOM (Document Object Model), což je objektová reprezentace HTML kódu.

2.1.3 DOM (Document Object Model)

DOM představuje objektovou reprezentaci XML nebo HTML dokumentu. Tato reprezentace umožňuje pomocí svého rozhraní přistupovat k obsahu dokumentu a dále jej dynamicky upravovat. Stejně jako většina webových technologií má i DOM specifikaci [14] a doporučení konsorcia W3. Právě tato doporučení a specifikace umožnila sjednocení rozhraní mezi jednotlivými prohlížeči, což se pozitivně projevilo zejména na stále menším množství *browser-specific* kódu, tedy kódu specifického pro jednotlivé prohlížeče. Jednotlivé elementy jsou v dokumentu pomocí DOM reprezentovány jako strom čímž lze definovat jejich hierarchii.

2.1.4 Jazyk JavaScript

JavaScript je objektově orientovaný jazyk nejčastěji využívaný na webových stránkých a aplikacích. Primárně je využívaný pro tvorbu interakcí s uživatelem a je tak hlavním prostředkem pro stále se zvyšující důraz na kvalitu uživatelských rozhraní.

Dříve byl JavaScript spouštěn pouze na straně klienta, nyní je už ale možné díky technologiím jako Node.js spouštět JavaScript i na serveru [3]. JavaScript se těší stále větší a větší popularitě a díky nástrojům jako React Native [2] je možné ho využít i pro tvorbu mobilních aplikací.

2.1.5 Jazyk PHP

PHP je imperativní, dynamicky typovaný a objektově orientovaný jazyk jehož vydání sahá do roku 1995. Za tuto dobu k sobě přitáhnul širokou komunitu vývojářů, kteří vytváří knihovny, nástroje a celé projekty postavené právě na tomto jazyce. Široká komunita je jednou z hlavních výhod tohoto jazyka.

Na poli jazyků pro tvorbu *backend* části webových aplikací má dnes PHP velkou konkurenci v podobě Javy, C#, Pythonu nebo Golangu. V období před vydáním PHP ve verzi 7, nebyl tento jazyk příliš oblíbený hlavně kvůli jeho rychlosti, která byla daleko za konkurencí. Po vydání verze 7 však podle benchmarků došlo k rapidnímu zrychlení zpracování a také ke zlepšení paměťové náročnosti. Podle benchmarku z webu gbksoft.com[1] u aplikace využívající redakční systém Wordpress, je PHP ve verzi 7 schopné zpracovat 86,66 požadavků za sekundu, zatímco PHP ve verzi 5.6 pouze 35,93 požadavků za sekundu.

Ačkoliv došlo ve vývoji PHP k výrazným zlepšením zejména v rychlosti a paměťové náročnosti, je nutné vzít v úvahu i slabší stránky tohoto jazyka. Jedním z hlavních faktorů ovlivňující volbu tohoto jazyka je potřeba paralelizace zpracování. PHP neumožňuje asynchronní zpracování požadavků, jelikož je každý požadavek zpracován sekvenčně a to má samozřejmě např. oproti Node.js negativní vliv na rychlost obsluhy požadavků.

[1] https://gbksoft.com/blog/php-5-vs-php-7-performance-comparison/#SzqEy

2.2 Existující nástroje pro detekci technologií

Aktuálně je dostupné velké množství nástrojů pro detekci webových technologií a některé z nich je možné využívat i bezplatně.

2.2.1 Aplikace BuiltWith

První nástrojem je webová stránka builtwith.com[2], která jednoduše na základě zadané adresy zjistí technologie použité na dané stránce. Nástroj je schopen rozpoznat technologie použité na serveru (např. Apache, Nginx, PHP), využití měřících nástrojů jako Google-Analytics, použití javaskriptových knihoven, využívání CDN (Content Delivery Network) a další.

Hlavní výhodou nástroje je zobrazení počtů stránek, které danou technologii používá jak celosvětově, tak i s ohledem na region.

2.2.2 Aplikace ChromeLibraryDetector

Druhým nástrojem je ChromeLibraryDetector, který je už techničtějšího rázu. Pro jeho použití je třeba ho nainstalovat jako doplněk do prohlížeče Google Chrome. Jde o nástroj, který pro detekci technologií využívá i v poslední době velmi populární nástroj Chrome Lighthouse, který slouží k vyhodnocení parametrů stránky jako například rychlost, bezpečnost nebo best practices.

Jelikož jde o open source nástroj, je v dostupném kódu [6] vidět jak detekce probíhá. Ačkoliv tento nástroj detekuje pouze javaskriptové techonologie, je velmi užitečný v kontextu mé práce, protože způsob, jakým nástroj detekci provádí, bude využit i v mé aplikaci.

2.2.3 Aplikace Wappalyzer

Třetí nástroj, který je velice podobný cílové aplikaci mé bakalářské práce, je webová aplikace Wappalyzer[3]. Detekuje redakční systémy, platformy pro elektronické obchody, knihovny pro tvorbu webových aplikací nebo marketingové nástroje. Důležitou vlastností aplikace je otevřenost jejího zdrojového kódu, který je volně dostupný pomocí služby GitHub.

I přes podobnost tohoto nástroje jsem se rozhodl jej nerozvíjet a zvolit vlastní řešení. Přídavná funkcionalita mé aplikace, v podobě výpisu zranitelností, vyžaduje specifická řešení některých problémů, která by mohlo být v některých případech složité implementovat jako nadstavbu již existujícího nástroje.

Aplikace Wappalyzer bude využita pro porovnání funkčnosti implementované aplikace v závěru práce, kdy bude sledována úspěšnost detekce daných technologií na vhodných webových stránkách. Ačkoliv detekuje mj. i technologie, které jsou v kontextu implementované aplikace nedůležité, je tento nástroj pro porovnání nejvhodnější.

2.3 Způsoby a principy detekce technologií

Tato část popisuje, jakými způsoby lze detekovat technologie použité na webových stránkách. Způsobů detekce je několik a také některé způsoby umožňují detekovat pouze určitou kategorii technologií. Není tedy jediný způsob detekce, který by byl schopen zjistit vše.

[2] https://builtwith.com/
[3] https://www.wappalyzer.com/

Místo toho je nutné pro komplexní detekci spojit více způsobů, aby byl výčet technologií co nejobsáhlejší a nejpřesnější.

2.3.1 Detekce z hlaviček HTTP zpráv

Prvním způsobem je detekce z hlaviček zpráv protokolu HTTP. Jak bylo popsáno v kapitole 2.1.1 o protokolu HTTP, velké množství informací na sebe může server prozradit v odpovědi na HTTP požadavek. Z odpovědi od serveru je například možné poznat jaký software webový server používá (Nginx, Apache, ...), jestli stránka využívá nějaký webový akcelerátor (např. Varnish) nebo je možné například pomocí hlavičky `X-Powered-By` zjistit, zda stránka využívá jazyk PHP a případně v jaké verzi. V některých případech může být v této hlavičce uveden i redakční systém, na kterém webová stránka běží (např. Craft CMS).

Dále je možné pro účely zpětné vazby ověřit, zda server posílá některé bezpečnostně důležité hlavičky, jako například `X-XSS-Protection` nebo `Strict-Transport-Security`.

2.3.2 Testování existence adres nebo vyhledávání v URI

Zajímavým způsobem je testování existence určitých adres daného webu a také vyhledávání v URI zdrojů, které stránka požaduje. Při testování existence adres můžeme zvolit adresu, kde očekáváme přihlášení do administrace webu. Při vyhledávání v URI poté vyhledáváme řetězce charakteristické pro jednotlivé technologie.

Testováním adresy s příhlášením do administrace, lze detekovat použití redakčních systémů. Zatímco u redakčního systému Wordpress[4] je přihlášení do administrace na adrese `/wp-login.php`, u redakčního systému Drupal[5] je toto přihlášení na adrese `/user/login`. Další oblíbený redakční systém Joomla[6] má přihlášení do administrace obvykle na adrese `/administrator`. Jako univerzální adresu pro přihlášení do administrace lze brát `/admin`, ta nám ale sama o sobě není schopná určit, o jaký redakční systém jde, na což je třeba využít i jiné způsoby detekce. Doplňujícím zdrojem informací k této části může být i obsah souboru na adrese `/robots.txt`, pomocí něhož lze zabránit vyhledávačům v indexaci konkrétních částí webu, jako je právě jeho administrace. Je vhodné doplnit, že tento způsob není vždy použitelný; zejména v případě, kdy tvůrce webu přesunul administraci na neobvyklou adresu nebo je dostupná pouze na neobvyklém portu (např. 8080).

Vyhledáváním v URI jednotlivých zdrojů lze zjistit jak využití redakčních systémů, tak i použití knihoven pro tvorbu uživatelských rozhraní. Web postavený na redakčním systému Wordpress může načítat zdroje jako JS nebo CSS soubory z adres, které obsahují řetězec `/wp-content/`. E-shop využívající redakční systém Magento[7] bude v případě, že nevyužívá CDN (Content Delivery Network), načítat zdroje z adresy obsahující řetězec `/mage/`. Stránka využívající redakční systém Drupal bude požadovat soubory z adresy obsahující řetězec `/sites/`. Další možností je také vyhledávání v samotném názvu souboru, kdy v případě použití knihovny Bootstrap[8] bude stránka načítat soubory s názvem `bootstrap.js` a `bootstrap.css` nebo jejich minifikované verze.

[4] https://wordpress.com/
[5] https://www.drupal.org/
[6] https://www.joomla.org/
[7] https://magento.com/
[8] https://getbootstrap.com/

2.3.3 Vyhledávání ve zdrojovém kódu

Dalším způsobem pro detekci technologií je stažení kódu a následné vyhledávání v jeho obsahu. Vyhledávat můžeme jak v HTML kódu dané stránky, tak i v jejím javaskriptovém kódu. V kódu můžeme vyhledávat jednoduchým způsobem pomocí podřetězců, nebo pomocí komplexnějších regulárních výrazů. ve většině případů však regulární výrazy využijeme až v případě, kdy potřebujeme extrahovat data ze zdrojového kódu.

Vyhledávání pomocí podřetězců

Jako nejlepší způsob detekce technologií můžeme s jistotou označit vyhledávání v javaskriptovém kódu dané stránky. Zajímavými řetězci v tomto ohledu mohou být funkce specifických názvů nebo často používané komentáře, například ke specifikování licence, ve které je daná technologie publikována. Kód knihovny React.js obvykle obsahuje v komentářích část `@license React`.

Komentáře obecně mohou být spolehlivým zdrojem informací o využitých technologiích, protože se vývojáři velmi často pomocí komentářů odkazují na externí dokumentaci dané technologie, takže komentář s odkazem `See https://redux.js.org/api-reference...` napovídá, že je na stránce využita knihovna Redux.js.

Extrakce verze technologie pomocí regulárních výrazů

U aplikace kromě detekce technologií chceme detekovat i verzi, ve které jsou dané technologie využity. Této funkcionality nelze dosáhnout obyčejným vyhledáváním podřetězců, ale je nutné využít regulární výrazy pomocí kterých jsme schopni extrahovat libovolná data z textových dokumentů, kterými zdrojové kódy bezpochyby jsou.

V kapitole 2.3.3 je uveden příklad ze zdrojového kódu technologie React.js, který obsahuje komentář s informací o licenci, ve které je daný produkt publikován. Narozdíl od předchozího příkladu nás bude zajímat celý řádek, který vypadá takto: `@license React v16.7.0`. Jak můžeme vidět, kromě názvu technologie tento řádek obsahuje i verzi, zbývá tedy tuto informaci z kódu extrahovat pomocí regulárního výrazu. Pro tento příklad můžeme verzi získat pomocí výrazu `\/\*\* @license React v(\d+\.?d+\.?\d+)`.

Tento způsob lze využít pro široké spektrum technologií, nicméně je nutné pro každou technologii vytvořit odpovídající regulární výraz. Bohužel kvůli závislosti tohoto způsobu na obsahu zdrojového kódu není aplikace schopna pružně reagovat na případné změny.

2.3.4 Detekce z window objektu dané stránky

Objekt `window` je vytvořen pro každé okno prohlížeče a v případě, že má stránka vloženou další stránku (`iframe`), je vytvořen tento objekt i pro každý tento `iframe`. Objekt `window` využívá i nástroj ChromeLibraryDetector zmíněný v kapitole 2.2.2.

Detekce technologií probíhá tak, že se v tomto objektu vyhledávájí podle názvu jednotlivé javaskriptové knihovny, které stránka vyžaduje. Po načtení stránky, která využívá například jQuery, tedy bude objekt `window` obsahovat i objekt s názvem `jQuery`. Takto lze v některých případech detekovat také již dříve zmíněné technologie jako React.js, Angular nebo Bootstrap.

Tento způsob detekce selhává v případech, kdy javaskriptová knihovna neobsahuje svoji referenci ve window objektu. Toto je závislé na způsobu implementace a typu dané knihovny.

2.3.5 Zpracování souborů pro správu závislostí

Způsob, který se od předchozích liší nejvíce, je zpracování souborů pro správu závislostí. Dnes je již standardem využívat při tvorbě SW projektů správce závislostí (správce balíčků), jakými jsou `npm`[9], `Composer`[10] nebo `yarn`[11].

Každý z těchto nástrojů vytvoří v projektu vlastní soubor (např. `composer.lock`), kde je uložen aktuální strom závislostí mezi jednotlivými balíčky a také jsou zde uchovány informace o tom, v jaké jsou verzi. Zpracováním těchto souborů tak můžeme získat podrobný soupis technologií použitých na daném projektu.

[9] https://www.npmjs.com/
[10] https://getcomposer.org/
[11] https://yarnpkg.com/

Kapitola 3

Databáze softwarových zranitelností

Druhá kapitola práce se zaměřuje na zranitelnosti softwaru, vysvětluje některé důležité pojmy s nimi související, představuje některé zdroje informací o softwarových zranitelnostech a okrajově představuje nejobvyklejší slabiny z pohledu webových stránek.

V kontextu standardů využívaných při práci se softwarovými zranitelnostmi je nutné odlišit dva pojmy - softwarová slabina a softwarová zranitelnost. I když se může na první pohled zdát, že jde o dva různé názvy pro to samé, není tomu tak.

3.1 Softwarová slabina

Softwarovou slabinou, myslíme jakoukoli chybu v softwaru, která může vést k jeho zranitelnosti. Softwarová slabina nemá vazbu na konkrétní produkt nebo program.

Ve formálním slovníku softwarových slabin CWE™(Common Weakness Enumeration) [10], můžeme najít pro odbornou veřejnost velmi dobře známé pojmy jako Cross-Site Scripting (XSS), SQL Injection nebo například ukládání dat v *plaintext* podobě. Tyto tři slabiny si nyní okrajově představíme.

3.1.1 Cross-Site Scripting (XSS)

XSS je aktuálně jednou z nejčastěji vyskytujících se slabin. V žebříčku slabin, vytvořeném organizací OWASP [11], se umístil XSS na sedmém místě. Jeho hlavní příčinou je neošetření uživatelského vstupu, čímž je umožněno útočníkovi podstrčit do webové stránky vlastní škodlivý kód.

S pomocí XSS lze provést útoky jako Session hijacking, což je získání identifikátoru sezení jiného uživatele a jeho následné využití při autentizaci, nebo phishing, kdy jsou do stránky vložena podvodná data a uživatel je s jejich pomocí obvykle vyzván k zadání citlivých údajů.

Proti této slabině se lze chránit důkladným *escapováním* vstupů, kdy má většina jazyků pro tento účel připraveny vestavěné funkce, a také zasíláním hlavičky `X-XSS-Protection`, která je podporována všemi moderními prohlížeči s výjimkou Firefoxu.

3.1.2 SQL Injection

SQL Injection je také slabina způsobená neošetřením vstupu, tentokrát však dochází k provedení vloženého škodlivého kódu na úrovni databáze. Díky ní může útočník získat data z databáze, či případně měnit nebo mazat její obsah.

I zde je ochranou důkladné *escapování* vstupů, či případné využívání tzv. Prepared Statements. Ty primárně slouží k zrychlení a zefektivnění opakujících se SQL dotazů; nicméně díky tomu, že jsou hodnoty parametrů přenášeny samostatně, není třeba aby byly ošetřeny.

3.1.3 Ukládání dat v čitelné podobě

Potenciálně velmi nebezpečnou slabinou je ukládání a následné uchovávání citlivých dat v čitelné podobě. Mezi nejdůležitejší citlivá data patří přihlašovací údaje (e-mail, už. jméno, heslo, ...), kdy v případě jejich úniku může útočník poškodit velké množství uživatelů.

Představme si situaci, kdy díky slabině SQL Injection útočník získá data z databáze některého webového portálu s možností přihlášení. ve chvíli, kdy jsou zejména hesla uživatelů uložena v čitelné podobě, tedy nezašifrována, nic útočníkovi nebrání v získání přístupu k cizímu účtu.

Této slabině se dá předcházet šifrováním citlivých údajů. Dnešní technologie poskytují velké množství šifrovacích algoritmů, které jsou vhodné právě například pro šifrování hesel. Aktuálně mezi nejpoužívanější algoritmy patři `bcrypt` a `Argon2i`, díky nimž je lámání hesel i za pomoci GPU z hlediska časové náročnosti v dnešní době téměř nemožné.

3.2 Softwarová zranitelnost

Softwarovou zranitelností myslíme chybu v softwaru, která může být útočníkem využita k převzetí kontroly nad systémem nebo získání citlivých informací, které jsou tímto systémem spravovány. Softwarová zranitelnost se již vztahuje ke konkrétnímu produktu nebo programu (v kontextu předchozí kapitoly můžeme použít i slovo technologii).

Stejně jako softwarové slabiny mají i softwarové zranitelnosti svůj formální seznam, v tomto případě jde o CVE®(Common Vulnerabilities and Exposures) [8]. Při pochopení rozdílu mezi slabinou a zranitelností softwaru je zřejmé, že tento seznam bude daleko obsáhlejší.

3.2.1 CVVS - Common Vulnerability Scoring System

CVSS[1] poskytuje možnost ohodnocení, o jak vážnou zranitelnost jde, na základě některých charakteristických znaků. Výsledkem ohodnocení je číslo v rozmezí 0-10, kdy 10 bodů znamená velice vážnou zranitelnost. Tento systém bodování pomáhá vývojovým týmům prioritizovat jednotlivé zranitelnosti s ohledem na jejich opravování.

Výpočet základního skóre (CVVS Base Score) je založen na hodnotě těchto vlastností:

- **Attack Vector** - hodnota je založená na kontextu, ze kterého je možné zranitelnost využít,
- **Attack Complexicity** - míra *práce*, potřebná k úspěšnému provedení útoku,
- **Priviliges Required** - míra práv, potřebných k provedení útoku,

[1] https://www.first.org/cvss/

- **User Interaction** - označuje potřebu vstupu od uživatele,
- **Scope** - hodnotu určuje možnost, ovlivnit zdroje mimo napadený systém,
- **Confidentiality Impact** - určuje do jaké míry jsou data v systému citlivá,
- **Integrity Impact** - specifikuje možnost ovlivnění důveryhodnosti a pravdivosti dat,
- **Availability Impact** - vliv na dostupnost služby.

Vhodným nástrojem pro výpočet skóre zranitelnosti je CVVS Calculator[2], který umožňuje výpočet nejen základního skóre, ale i kompletního skóre, na základě velkého množství zvolených parametrů.

3.3 Existující databáze softwarových zranitelností

Tato část je věnována popisu několika zdrojů poskytujících přehled softwarových zranitelností, kdy ve všech případech jde o webové portály, přičemž některé z nich mají vytvořené i soubory dat, pomocí kterých lze zranitelnosti snadno zpracovávat. Také bude vysvětlen způsob jakým je vytvářeno skóre zranitelností, které pomáhá určit vážnost dané zranitelnosti.

3.3.1 CVE - Common Vulnerabilities and Exposures

Hlavním celosvětovým zdrojem informací o zranitelnostech je bezesporu CVE[3]. Definuje standardy pro shromažďování a popisování zranitelností a je tak vhodným *zdrojem pravdy* pro další databáze zranitelností.

V databázi CVE má každá zranitelnost své unikátní číslo, které je tvořeno rokem přidání do databáze a pořadovým číslem (např. `CVE-2018-16845`). Díky tomuto identifikátoru, lze velmi dobře odkazovat na detail zranitelnosti a také v případě potřeby ověřit, zda zranitelnost skutečně existuje.

3.3.2 NVD - National Vulnerability Database

NVD[4] je webové datové úložiště vlády Spojených států Amerických pro práci se softwarovými zranitelnostmi. Jde o rozsáhlý webový portál, který poskytuje jak přímé vyhledávání na základě klíčových slov, tak i možnost stažení pravidelně aktualizovaných kanálů (*feedů*) ve formátu JSON nebo XML. Jejich výhodou je striktně standardem definovaná podoba, címž jsou vhodné pro další zpracování.

Jak organizace MITRE sama deklaruje [9], NVD je plně synchronizována s databází CVE, i když jde o dva oddělené projekty. Při jakékoli úpravě v CVE se tato úprava okamžitě promítne i do databáze NVD.

NVD poskytuje dva zdroje dat, které v aplikaci budou využity. Prvním je slovník technologií (NVD Product Dictionary), obsahující jednotlivé verze technologií, a také seznamy zranitelností, které lze po zpracování spojit s jednotlivými verzemi technologií.

[2] https://www.first.org/cvss/calculator/3.0
[3] https://cve.mitre.org/
[4] https://nvd.nist.gov/

V kontextu mé práce je při zpracování velmi důležité důležité určení produktů a verzí, kterých se určitá zranitelnost týká. V XML *feedu* z NVD je tato vlastnost komplexně specifikována v elementu `vuln:vulnerable-configuration`. ve *feedu* je podporován i komplexnější způsob určení produktů pomocí logických spojek `AND` a `OR`.

3.3.3 Snyk Vulnerability Database

Kvalitní databází softwarových zranitelností je i Snyk Vulnerability Database [5]. Umožňuje vyhledávání zranitelností na základě názvu technologie a nebo pomocí čísla zranitelnosti z databáze CVE. Narozdíl od předchozí databáze NVD, Snyk neposkytuje veřejně dostupné *feedy* pro další práci se zranitelnostmi. Databázi používá v první řadě samotný Snyk u svého řešení pro vyhledávání a opravu zranitelností v softwarových projektech.

Specifickou vlastnotí této databáze je rozdělení zranitelností do kategorií podle toho, jakým systémem jsou technologie spravovány. PHP knihovny jsou tedy v kategorii `Composer` a zranitelnosti webových serverů jako nginx jsou v kategorii `Linux`. Toto rozdělení může být pro uživatele matoucí a není tedy přiliš vhodné.

[5] https://snyk.io/vuln/

Kapitola 4

Zpracování webových stránek

Různé způsoby detekce techonologií zmíněné v kapitole 2.3, vyžadují různé způsoby zpracování webových stránek. Tato kapitola se věnuje možnostem a knihovnám, které se dají pro zpracování využít.

4.1 Vykreslení v prohlížeči

Vykreslení v prohlížeči slouží zejména pro uživatele, který chce procházet danou webovou stránku. Tento způsob zpracování se dá velmi dobře využít při akceptačním testování, kdy je možné pomocí API pro automatizaci ovládání prohlížečů ovládat danou stránku stejným způsobem, jakým to může dělat uživatel.

Při akceptačním testování lze provádět testy z pohledu uživatele, proto akceptační testy představují poslední úroveň v hierarchii jednotlivých způsobů testování. Největší výhodou například oproti funkcionálnímu testování, je v případě webových aplikací možnost testovat i funkcionality závislé na JavaScriptu a provádět interakce přímo s uživatelským rozhraním dané stránky.

Právě kvůli možnosti provedení javaskriptového kódu je tento způsob zpracování vhodný pro způsob detekce technologií z `window` objektu, který je popsán v kapitole 2.3.4. Díky API je možné do vykreslené stránky injektovat vlastní javaskriptový kód, kterým lze získat obsah objektu `window` a dále s ním pracovat.

4.1.1 WebDriver

WebDriver je protokol nezávislý na platformě a programovacím jazyce umožňující vzdáleně ovládat webovou stránku. Poskytuje velké množství funkcí, pomocí nichž je možné procházet a měnit elementy DOM stromu nebo interagovat se stránkou jako v případě ovládání uživatelem. Protokol je specifikován v doporučení konsorcia W3 [15].

WebDriver je pro jednotlivé prohlížeče dostupný pomocí jejich implementací tohoto protokolu. Pro prohlížeč Google Chrome je to chromedriver, pro Firefox jde o Gecko Driver. Implementace WebDriveru jsou ale dostupné i pro Safari, Operu nebo Android. Pokud chceme WebDriver využít například v jazyce PHP, je dostupná knihovna php-webdriver[1] od společnosti Facebook Inc.

[1] https://github.com/facebook/php-webdriver

4.1.2 Bezhlavičkový prohlížeč Zombie.js

Alternativou k využití protokolu WebDriver a nástrojů jako chromedriver je bezhlavičkový prohlížeč Zombie.js[2]. Vzhledem k tomu, že jde o odlehčenou verzi prohlížeče, implementovanou v jazyce JavaScript, je vykreslení stránky velmi rychlé.

Ačkoliv není deklarováno, že Zombie.js implementuje protokol WebDriver, syntaxe je velice podobná a to zjednodušuje případný přechod mezi konkurenčními nástroji. Pro svůj běh využívá prohlížeč Zombie.js prostředí javaskriptového serveru Node.js.

4.2 Procházení DOM stromu

Elementy dokumentu lze reprezentovat ve formě stromu, nicméně pro jejich procházení je nutné využít další techniky. Nejobvyklejší je pro procházení elementů využití výrazů používajících CSS selektory nebo využití jazyka XPath.

4.2.1 Procházení pomocí CSS

Intuitivním způsobem pro proházení HTML elementů může být využití CSS selektorů. Jde o stejný způsob jako v případě definic stylů pomocí CSS. Elementy je možné specifikovat pomocí identifikátoru (např. `#container`), pomocí třídy (např. `.btn`) nebo podle názvu elementu a případně i hodnoty atributu (např. `input[type="checkbox"]`).

Nevýhodou tohoto způsobu procházení je absence složitějších výrazů, které mohou být v některých případech užitečné. V rámci HTML dokumentů je však ve většině případů tento způsob dostačující.

```
#wrapper .container form input[type="radio"]
```

Výpis 4.1: Ukázka výrazu pro selekci pomocí CSS pravidel

4.2.2 XPath

XPath je jazyk pro procházení elementů XML a HTML elementů. Lze pomocí něj vybírat jak jednotlivé elementy, tak i jejich skupiny. ve výrazech jazyka XPath je možné využívat i pokročilejší funkce jako například funkce po porovnávání numerických hodnot nebo časových údajů. Stejně jako DOM je i XPath součástí definice konsorcia W3 [16].

Základní konstrukce jsou tvořeny počátečním symbolem `/` pro vyhledávání v kořenu dokumentu nebo `//` pro vyhledávání v aktuálním elementu. Dále je možné využívat navigaci jako v procházení adresářové struktury v příkazové řádce. Pro selekci atributů lze využívat znak `@`.

```
vuln:item/vuln:cvss/cvss:base_metrics/cvss:score
```

Výpis 4.2: Ukázka výrazu pro selekci pomocí XPath

4.3 Stažení kódu stránky

Nejjednodušší a nejlepší způsob detekce je vyhledávání typických vzorků v kódu dané webové stránky, zmíněné v kapitole 2.3.3. Aby bylo možné v kódu vyhledávat, je nutné získat všechny zdrojové soubory dané stránky, konkrétně HTML a javaskriptový kód. Po

[2] http://zombie.js.org/

stažení těchto souborů je možné pomocí jednoduchého vyhledávání řetězců nebo za využití regulárních výrazů vyhledávat konkrétní znaky jednotlivých technologií.

V minulosti, kdy nehrál v oboru webových technologií JavaScript tak zásadní roli, by stačilo pro získání *offline kopie* použít některý z nástrojů příkazové řádky jako `wget`[3] nebo `cURL`[4]. Bohužel tyto nástroje nejsou schopny provádět javaskriptový kód a nemohou stáhnout zdroje, které jsou na stránku přidávány dynamicky. Dynamické přidávání kódu je dnes využíváno zejména kvůli možnostem poskytnout více verzí kódu v závislosti na klientovi, který ke stránce přistupuje.

Z tohoto důvodu bude v kontextu mé práce nutné pro tento účel využít zpracování pomocí vykreslení v prohlížeči, které je popsáno v kapitole 4.1.

[3] https://www.gnu.org/software/wget/
[4] https://curl.haxx.se/

Kapitola 5

Návrh aplikace

Tato kapitola bude věnována základnímu návrhu architektury apikace, návrhu schématu databáze a také volbě implementačních technologií. V neposlední řadě jsou zde také uvedeny požadované funkcionality na straně uživatelského rozhraní.

5.1 Základní architektura

Hned zpočátku lze architekturu aplikace rozdělit na dvě části - *backend* a *frontend*. Každá z těchto částí je zodpovědná za jiné úkoly a bude implementována pomocí odlišných technologií. Pro správnou koordinaci těchto částí je nutné zvolit vhodný způsob, kterým spolu budou komunikovat.

5.1.1 Komunikační vrstva (JSON API)

Ze základního rozdělení aplikace na dvě části je jasné, že je nezbytné zvolit vhodný způsob komunikace mezi backendovou a frontendovou částí aplikace. Vzhledem k plánu využít pro implementaci uživatelského rozhraní knihovnu pro tvorbu uživatelských rozhraní a také aktuálním trendům ve vývoji webových aplikací, se komunikace pomocí JSON API[1] jeví jako nejlepší volba. Implementace JSON API má velké výhody i z pohledu testování, kdy se dá API testovat odděleně a bez uživatelského rozhraní.

Při implementaci je nutné zvolit i vhodný formát dat v odpovědi. S přihlédnutím ke skutečnosti, že implementačním jazykem pro uživatelské rozhraní bude JavaScript, je formát JSON nejlogičtější volbou např. před XML. Práce s daty ve formátu JSON v jazyce JavaScript nevyžaduje explicitní převádění nebo zpracování. Data jsou reprezentována ve formě objektů, jak je v tomto jazyce obvyklé.

API endpointy

API endpoint neboli koncový bod API je komunikační kanál identifikovaný pomocí URL. V aplikaci bude několik endpointů, přičemž každý z nich bude mít svůj specifický účel. Kromě endpointů pro získaní zranitelností bude nutné implementovat i endpointy pro získání výsledku detekce, nebo pro úpravy tohoto výsledku.

[1] https://jsonapi.org/

5.1.2 Backend aplikace

Backend aplikace, implementovaný pomocí vhodného frameworku, bude tvořen skupinami vhodně vytvořených služeb (services), které budou plnit svůj účel. Tyto služby budou poté využívány v kontrolerech, které budou obsluhovat jednotlivé požadavky.

Backend aplikace bude tvořen importem dat do databáze; službami, které budou využívány pro provedení detekce; a také kontrolery, které budou obsluhovat jednotlivé požadavky.

5.1.3 Frontend aplikace

V této části bude implementováno uživatelské rozhraní aplikace. Kromě javaskriptové implementace bude ve frontendové části aplikace nutné implementovat i HTML kód aplikace a také definici stylů stránky, přičemž bude vhodné zvolit knihovnu pro tvorbu uživatelských rozhraní.

Pro správu závislostí v této části implementace je obvyklé využít správce balíčků. Tento nástroj zjednodušuje přidávání jednotlivých knihoven a umožňuje nastavení nejrůznějších úkolů při kompilaci kódu frontend části aplikace. Díky tomuto nástroji lze také velmi jednoduše zlepšit přenositelnost dané aplikace a také do aplikace přináší přehled použitých nástrojů a knihoven.

5.2 Knihovny a frameworky v jazyce JavaScript

Vhodným jazykem pro implementaci uživatelských rozhraní je v kapitole 2.1.4 popsaný jazyk JavaScript. V dnešní době existuje široká škála knihoven a frameworků, jejichž rychlý rozvoj přispívá k rychlému rozvoji celé javaskriptové komunity. Součástí tohoto rozvoje byl i vznik jazyků, které z jazyka JavaScript vycházejí a vylepšují ho, jako např. TypeScript[2].

5.2.1 React.js

Aktuálním trendem mezi knihovnami pro tvorbu uživatelských rozhraní je React.js[3]. Jak už název napovídá, jde o knihovnu v jazyce JavaScript. Princip fungování spočívá v komponentovém přístupu, kdy jsou tyto komponenty za využití jejich vnitřního stavu prezentovány uživateli jako elementy webové stránky.

U React.js se zapojuje do komunikace API rozhraní vytvořené v backend části webu. Jednotlivé komponenty získávají z API endpointů svůj výchozí stav a případně zasílají pomocí API požadavky na uložení při změně svého interního stavu.

U javaskriptových knihoven obecně je nutné nezapomínat na skutečnost, že k vytvoření HTML kódu dochází ve výchozím nastavení až na klientovi, tedy v prohlížeči uživatele. To může mít v některých případech negativní dopad zejména na SEO takto implementovaných aplikací. Ačkoliv došlo v této oblasti k velkému posunu, stále mají některé vyhledávače problémy s indexací stránek implementovaných pomocí javaskriptu. [4]

Řešením toho problému nezávisle na vyhledávači je tzv. server-side rendering. Při tomto přístupu je odpověď generována na serveru a aplikace je tak přístupnější pro všechny vyhledávače. V kontextu mé aplikace však indexace vyhledávači není nutná a proto bude dostačující generování HTML kódu na straně klienta.

[2] https://www.typescriptlang.org/
[3] https://reactjs.org/

Virtual DOM

V kapitole 2.1.3 je představen DOM jako technologie standardu W3C, která je využita ve všech webových prohlížečích pro objektovou reprezentaci dané webové stránky. DOM je možné upravovat a tím tedy dynamicky ovlivňovat podobu stránky bez nutnosti jejího kompletního znovunačtení. Tyto úpravy jsou však nákladné na výkon a u složitějších uživatelských rozhraní s vysokým počtem elementů mohou tyto úpravy zpomalit celou stránku. Důvodem zpomalení je skutečnost, že jsou na každou takovou úpravu napojeny interní činnosti prohlížeče jako přepočet CSS pravidel a úprava souřadnic elementů. [1]

Virtual DOM je oddělenou reprezentací DOM, která je uložena v paměti. Zjednodušeně jde o vylepšenou kopii reálného DOM objektu. Knihovna React.js díky této kopii může pomocí pokročilých algoritmů porovnávat reálný DOM s Virtual DOMem a zjistit tak minimální potřebné množství úprav v DOMu reálném. Tento proces má ve výchozím stavu časovou složitost $O(n^3)$, ovšem React.js využívá heuristický přístup, kdy některé použité předpoklady snižují tuto složitost na $O(n)$ [7].

5.2.2 Angular

Javaskriptový framework Angular[4] poskytuje narozdíl od knihovny React.js, který reprezentuje v architektuře MVC pouze View, i části Model a Controller. Dalším rozdílem mezi React.js a frameworkem Angular je také odlišný přístup k úpravám DOM objektu. Zatímco React.js využívá v kapitole 5.2.1 popsaný Virtual DOM, Angular pracuje s reálným DOM objektem. To má u velmi komplexních uživatelských rozhraní negativní vliv na rychlost.

Rozdílný je také způsob předávání dat z části Model do View. V React.js je použit tzv. *one-way data binding*, u frameworku Angular je využit *two-way data binding*.

5.3 Dostupné PHP frameworky

Předností jazyka PHP je mj. existence několika frameworků pro tvorbu webových aplikací, jejichž využití by již mělo být v dnešní době standardem. Frameworky pro tvorbu aplikací v jazyce PHP částečně odstiňují vývojáře od opakujících se problémů, na které může při tvorbě webových aplikací narazit, a zároveň se za něj starají o důležité části jako bezpečnost nebo rychlost. Pro rychlost aplikace je však nutné dodržovat postupy a způsoby implementace, které nebudou mít na rychlost frameworku negativní vliv.

5.3.1 Symfony Framework

Symfony Framework[5] je PHP framework pro tvorbu webových aplikací, který kombinuje přehlednost, jednoduchost a modulárnost. Mezi hlavní výhody tohoto frameworku patří mj. i v dnešním softwarovém vývoji hojně využívaná architektura MVC (Model-View-Controller).

Do webových projektů ssebou přináší Symfony Framework i systém pro správu závislostí, kterým je Composer[6]. Umožňuje přidávání velkého množství knihoven a nástrojů, při zachování přehlednosti a přenositelnosti.

Mezi přednosti tohoto frameworku patří zejména kvalitní dokumentace, a také široká komunita. Další výhodou frameworku Symfony, například oproti Nette Frameworku, je

[4] https://angular.io/
[5] https://symfony.com/
[6] https://getcomposer.org/

skutečnost, že za vývojem stojí komerční společnost SensioLabs. Díky tomu je rozvoj jednotlivých částí frameworku velmi rychlý. Právě kvůli absenci celosvětové komunity jsem se rozhodl z průzkumu Nette Framework vynechat a zaměřit se na celosvětově známé PHP frameworky.

Symfony Flex

Symfony Flex[7] představuje nový způsob tvorby webových aplikací v Symfony Frameworku. Doplněk Symfony Flex transformuje framework v počátečním stavu na pouhou kostru aplikace a neobsahuje tedy některé obvyklé knihovny a doplňky jako v případě předešlých verzí. Výhodou Symfony Flex je zejména možnost instalovat do aplikace pouze závislosti, které v aplikaci reálně používá.

Zajímavou změnou je také rozdílný přístup k nastavování konfiguračních hodnot. Symfony Flex využívá pro konfigurační hodnoty tzv. *environment variables*. Toto řešení umožňuje pohodlné nastavení aplikace na nejrůznějších prostředích.

5.3.2 Laravel

Laravel[8] představuje v aktuální době vhodnou alternativu k Symfony Frameworku. Není však vhodný pro velmi komplexní projekty, u kterých se doporučuje využít Symfony Framework, ačkoliv koncept MVC využívají oba tyto frameworky.

Narozdíl od Symfony Frameworku, u Laravelu neprobíhá cachování zdrojového kódu. Dochází tedy pouze ke cachování šablon což by mohlo naznačovat rychlostní deficit Laravelu. V benchmarku porovnávaných frameworků[9] byl však Laravel vyhodnocen jako lepší možnost. Stejně jako Symfony Framework i Laravel využívá pro správu závislostí Composer.

5.4 Volba konceptů a implementačních technologií

Na základě popsané architektury je nutné zvolit konkrétní technologie a koncepty pro implementaci aplikace. Pro backend i frontend část aplikace jsou zvoleny samostatné technologie využivající jiné programovací jazyky. Jedním z kritérií výběru však byla motivace nevyužívat větší počet programovacích jazyků. Na základě zvolených programovacích jazyků je třeba zvolit vhodné knihovny a frameworky a také je důležité vyhodnotit, které části aplikace bude nutné implementovat bez knihoven.

5.4.1 Koncept MVC (Model-View-Controller)

Mezi nejpoužívanější architektury kódu nejen webových aplikací, ale aplikací obecně, je architektura MVC. Jednotlivé znaky v názvu představují části aplikace s určitými oddělenými odpovědnostmi, přičemž jsou tyto mezi sebou propojeny, aby mohlo docházet k jejich vzájemné komunikaci a kooperaci. Tato architektura napomáhá i při kolektivním vývoji softwaru, kdy jsou v ideálním případě části aplikace rozděleny do většího množství samostatných souboru a adresářů.

První částí architektury MVC je Model jenž představuje zdroj dat, která jsou používána v aplikaci. Jako Model můžeme uvažovat např. MySQL databázi. Vstupem pro Model je požadavek na získání dat, s určitými parametry a výstupem jsou požadovaná data.

[7] https://symfony.com/doc/current/setup/flex.html
[8] https://laravel.com/
[9] https://thinkmobiles.com/blog/symfony-vs-laravel/

Další částí architektury MVC je Pohled (View), které slouží k zobrazování dat z Modelu. Pohled je obvykle tvořen např. HTML šablonami, které implementují uživatelské rozhraní aplikace. Díky Pohledu tak může uživatel data z Modelu vidět a dále s nimi pracovat. Platí, že jedna informace z Modelu, může být zobrazována pomocí několika různých Pohledů.

Spojovací částí mezi Modelem a Pohledem je Kontroler (Controller). Jak bylo zmíněno výše, Model a Pohled jsou oddělené části, a proto je nutné mít v aplikaci část, které bude obstarávat komunikaci mezi těmito dvěma částmi, přičemž by Kontroler neměl obsahovat logiku pro práci s daty. Hlavním úkolem je transformovat požadavek z Pohledu pro Model a data z Modelu pro Pohled.

5.4.2 Koncept ORM (Object-relational mapping)

V dnešním vývoji webových aplikací jsou hojně využívané nástroje pro objektově-relační mapování. Tyto nástroje odstiňují vývojáře od přímé práce s databází a umožňují reprezentaci dat z databáze pomocí objektů daného implementačního jazyka. Vzhledem k tomu, že většina PHP frameworků implicitně počítá s využitím ORM, není důvod se tomuto přístupu vyhýbat.

V aplikaci je nutné definovat skupinu tříd (entit), kdy jsou pomocí anotování specifikovány vlastnosti jednotlivých atributů tříd vzhledem k jejich podobě v databázi. Takto definované třídy jsou zpracovány a převedeny do formátu definice schématu databáze. V případě, že je databáze typu MySQL, je výstupem soubor MySQL dotazů pro vytvoření schématu dle specifikovaných nastavení. Při práci s daty poté pracujeme s řádky daných tabulek jako s objekty jednotlivých tříd, které reprezentují jednotlivé entity.

Nevýhodou použití ORM je zpomalení oproti přímému přístupu k výsledkům dotazů. U takového přístupu nedochází k tzv. hydrataci (hydration), tedy procesu generování objektů z výstupu z databáze a jejich plnění daty. Tento proces je poměrně náročný a proto v situacích kdy potřebujeme zpracovávat velké množství dat a zároveň nevyžadujeme výhody, které práce s objekty přináší, je lepší přistupovat přímo k výsledkům dotazů z databáze [12]. V takovém případě jsou obvykle výsledky reprezentovány jako asociativní pole.

5.4.3 PHP framework

I přes některé nedostatky jazyka PHP, jako dynamické typování, jsem zejména kvůli široké komunitě a existujícím knihovnám zvolil tento jazyk. S jazykem PHP mám velké množství zkušeností a při jeho využití by nemělo být komplikované dosáhnout potřebných funkcionalit.

Pro usnadnění a urychlení procesu vývoje aplikace jsem zvolil Symfony Framework popsaný v kapitole 5.3.1. Důvodem je zejména moje zkušenost s tvorbou aplikací v tomto frameworku a také snadná integrace knihovny pro tvorbu uživatelského rozhraní, díky knihovně Webpack Encore[10].

5.4.4 Prohlížeč pro vykreslení stránky

Na základě způsobů detekce uvedených v kapitole 2.3 je nutné v backend části aplikace využít knihovnu, která bude komunikovat s prohlížečem, ve kterém dojde k vykreslení stránky pro níž má být detekce provedena.

Ačkoliv se z počátku zdála být knihovna chromedriver spolu s vykreslením stránky v prohlížeči Google Chrome dobrou volbou, přešel jsem během dokončování implementace

[10] https://symfony.com/doc/current/frontend.html

na prohlížeč Zombie.js. Hlavním důvodem pro tuto změnu byla velmi dlouhá doba detekce, kterou se i přes různé kombinace nastavení knihovny chromedriver nepodařilo zkrátit.

5.4.5 Knihovna pro tvorbu uživatelského rozhraní

Trendem v dnešním vývoji aplikací je využití knihoven pro tvorbu uživatelského rozhraní. Možnost vybírat mezi několika knihovnami poskytuje příležitost zvolit tu nejvhodnější pro danou aplikaci a účel. U některých aplikací je nutné klást vysoký důraz na velikost zdrojového kódu knihovny, jinde může být rozhodující způsob jakým probíhá přenos dat mezi částmi Model a View nebo bude rozhodovat složitost uživatelského rozhraní.

Vzhledem k relativní jednoduchosti uživatelského rozhraní, kvalitě dokumentace, budoucímu využití v praxi a také přívětivosti v rámci zakomponování do aplikace, jsem zvolil knihovnu React.js popsanou v kapitole 5.2.1.

5.4.6 Webový server

Pro správné fungování aplikace na produkčním prostředí je nutné zvolit vhodný software pro webový server. Aktuálně dává smysl uvažovat pouze Apache[11] a nginx[12]. Důležitým měřítkem pro výběr je opět rychlost a jednoduchost napojení na další zvolené technologie, zejména PHP. Podle jednoho z benchmarků je s ohledem na počet zpracovaných požadavků nginx výrazně lepší než Apache, kdy je přibližně 2,5× rychlejší při zátěži 1000 současných připojení [13]. Tomuto výsledku odpovídá i trend, kde se převaha Apache v počtu využití s postupem času snižuje. Dále v textu tedy již budu popisovat zvolený webový server nginx.

Konfigurace

Aby byla webová aplikace dostupná na požadované URL, je nutné provést nastavení nginx serveru. Pro správné nastavení je třeba pochopit princip počátečního zpracování požadavku webovým serverem. Po zadání adresy do prohlížeče je klientem odeslán požadavek na tuto adresu. Dojde k překladu z doménového jména na IP adresu a dojde ke spojení se serverem s danou IP adresou. Pokud je na serveru spuštěn webový server, na základě URL požadavku zjistí zda je pro danou doménu vytvořena konfigurace. Kromě konfigurace domény je třeba určit také *Document root*, tedy kořenový adresář se zdrojovým kódem aplikace. Obvykle je vstupním bodem do aplikace soubor `index.php`.

FastCGI a PHP-FPM

V situaci, kdy zvolíme PHP jako programovací jazyk, je nutné umožnit komunikaci mezi webovým serverem a interpretem kódu naší aplikace. Pro komunikaci mezi těmito dvěma body obvykle využíváme protokol CGI (Common gateway interface), konkrétněji jeho aktualizovanou verzi FastCGI.

PHP-FPM je implementací protokolu FastCGI pro jazyk PHP. Je určen zejména pro aplikace s vysokou mírou zátěže, nicméně kombinace PHP, PHP-FPM a nginx je vhodnou volbou pro menší i větší webové aplikace nezávisle na míře jejich zátěže.

[11] https://httpd.apache.org/
[12] https://www.nginx.com/

5.5 Návrh schématu databáze

Pro většinu aplikací potřebujeme zajistit trvalost určitých dat tak, aby byly dostupné i po vypnutí dané aplikace či systému. U aplikace pro detekci technologií a vypsání zranitelností tomu není jinak.

V kapitole 5.4.2 popsané ORM definuje data vystupující v aplikaci, u nichž potřebujeme aby byly perzistentní, jako entity. Již z hlavních funkcionalit aplikace je jasné, že v databázi budeme uchovávat technologie a jejich zranitelnosti. Kromě těchto entit však budeme potřebovat uložit i jednotlivé provedené detekce s jejich výsledky.

5.5.1 Technologie a jejich verze

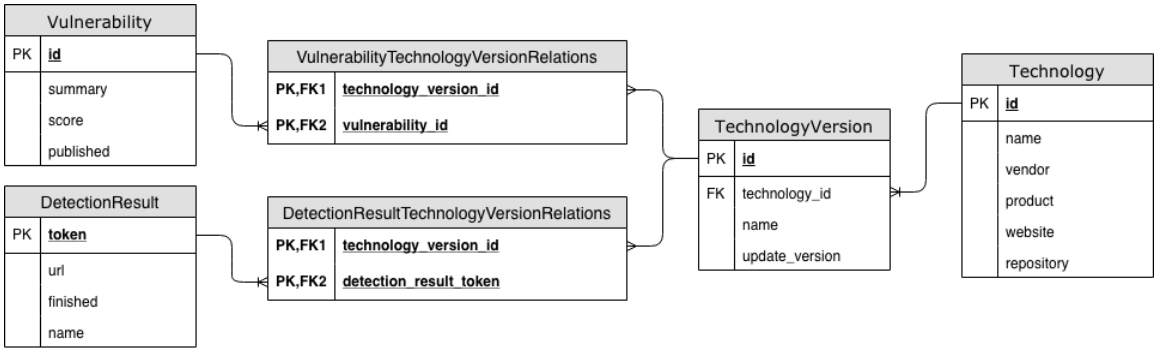
Technologie je entita reprezentující SW produkt, který je možné detekovat. Ke každé technologii potřebujeme uchovávat název produktu a také autora. Tyto dva atributy tvoří i unikátní identifikátor každé technologie. Vzhledem k tomu, že je aplikace schopna detekovat v některých případech i použitou verzi dané technologie, musíme v databázi uchovávat i jednotlivé verze technologií. Kromě výchozí verze, která bude využita v případě, že nebyla detekována konkrétní verze, budou uloženy všechny verze včetně vývojových verzí (např. alpha, beta, release candidate, ...), které získáme při importu dat. Každá technologie má jednu nebo více verzí, půjde tedy o vztah typu 1:N. V databázi i aplikaci bude entita Technologie nazvána `Technology`. Pro entitu reprezentující verzi technologie, použijeme název `TechnologyVersion`. Tabulky a vztah mezi nimi je znázorněn na obrázku 5.1.

5.5.2 Zranitelnosti technologií

U zranitelností potřebujeme v databázi uchovávat identifikátor CVE zranitelnosti, popis, skóre závažnosti a datum její publikace. Mezi takto uloženými zranitelnostmi je nutné vytvořit vazbu na konkrétní technologie. Přesněji na konkrétní verzi technologie. Tabulka `Vulnerability` tedy nebude ve vztahu s tabulkou `Technology` ale s tabulkou `TechnologyVersion`. Několik verzí technologie může mít více zranitelností, půjde tedy o vztah typu M:N a v databázi bude tento vztah uchováván pomocí vazební tabulky `VulnerabilityTechnologyVersionRelations`. Zmíněné tabulky a jejich vztahy jsou zakresleny na obrázku 5.1.

5.5.3 Záznam o detekci

Po provedení detekce potřebujeme uchovat její výsledek, a ten následně prezentovat uživateli. Požadavek uživatele na zobrazení výstupu detekce může být opakovaný. V záznamu o detekci potřebujeme uložit URL webové stránky, pro kterou byla detekce provedena, unikátní identifikátor dané detekce a také informaci o tom, zda byla detekce dokončena. Informaci, zda byla detekce dokončena potřebujeme kvůli asynchronnímu přístupu, který je nutný pro vyhnutí se blokujícímu načítání stránky s výsledkem detekce. Záznam o detekci bude uložen v tabulce `DetectionResult` a pro uchování vazby M:N, která platí pro vztah mezi záznamem o detekci a detekované verze technologií, bude využita tabulka `DetectionResultTechnologyVersionRelations`. Tyto tabulky a vztahy jsou znázorněny na obrázku 5.1.



Obrázek 5.1: ER diagram schématu aplikační databáze

5.6 Funkcionality uživatelského rozhraní

Po provedené detekci je uživatel pomocí užiatelského rozhraní informován o jejím výsledku. Jak bylo zmíněno, detekce nemusí být vždy přesná a některé technologie tak mohou zůstat při detekci opomenuty. Další technologie už z podstaty jejich využití detekovat nejde. Kvůli tomu je nutné přidat do aplikace funkcionality, které umožní uživateli dodatečně upravit výsledek detekce tak, aby odpovídal reálnému stavu.

5.6.1 Přidávání a odstraňování technologií

Funkcionalita přidávání a odstraňování technologií umožní uživateli doplnit do výsledku detekce technologie, které nebyly detekovány. Zároveň může dojít k situaci, že technologie, která se objevila ve výsledku detekce na webové stránce reálně použita není, a uživatel tak bude moci tuto technologii odstranit.

Aby bylo přidávání technologií intuitivní, bude nezbytné umožnit uživateli vyhledávání v technologiích. V této fázi lze do aplikace zapojit i statistiky a uživateli tak přednostně nabízet technologie, které jsou detekovány nebo dodatečně přidávány nejčastěji.

5.6.2 Úprava nebo zadání verze

Ani extrakce verze použité technologie nemusí být přesná, a tak musí být uživateli umožněno tuto verzi zadat nebo změnit, pokud byla detekována špatně. Tato funkcionalita bude mít vliv i na obsah výpisu zranitelností dané technologie, specifikací verze tedy dojde i k zpřesnění výpisu zranitelností.

S ohledem na velký počet verzí jednotlivých technologií je třeba, aby uživatel mohl ve verzích vyhledávat, ale zároveň je viděl vhodně seřazeny.

5.6.3 Sdílení výsledků detekce

Jedním z hlavních případů užití aplikace je situace, kdy člen vývojářského týmu spustí na určité stránce detekci a její výsledky chce sdílet i se zbytkem týmu. V takovém případě se dá využít jednoznačný identifikátor výsledku detekce.

Takový identifikátor by měl splňovat dvě podmínky. Neměl by být jednoduše uhodnutelný a také by mělo jít o náhodně vygenerovaný řetězec, tzv. `token`. Jelikož bude uživateli umožněno přidávat do výsledku detekce technologie, které se nepodařilo detekovat, může

výsledek o detekci obsahovat citlivá data, a není tak vhodné, aby byly jednoduše dohledatelné.

5.6.4 Alternativní specifikace technologií

Alternativou k detekci technologií je manuální specifikace technologií. Tato funkcionalita bude cílit na uživatele, kteří se teprve chystají webovou stránku implementovat a chtějí zjistit, které technologie by bylo vhodné použít. Díky aplikaci budou schopni zjistit zda technologie, které chtějí použít obsahují zranitelnosti, a na základě toho mohou zvolit bezpečnější alternativu.

Vzhledem k tomu, že půjde o dodatečnou funkcionalitu aplikace, není vhodné, aby přitahovala příliš velkou pozornost na úkor primární funkcionality, kterou je provedení detekce.

Kapitola 6

Implementace aplikace

V této kapitole jsou popsány zajímavé části implementace navržené aplikace. Primárně je kapitola věnována implementaci importu dat a s tím související transformaci, implementaci detekce technologií a také zajímavým částem implementace uživatelského rozhraní.

6.1 Import dat

Funkcionality aplikace jsou založeny na datech z externích zdrojů. Nabízela se možnost k těmto datům přistupovat přímo z jejich vzdáleného úložiště, nicméně toto řešení se s ohledem na rychlost a velmi omezené možnosti změn v získaných datech nedalo v navržené aplikaci využít. Proto bylo nutné implementovat logiku pro import dat z externích zdrojů.

6.1.1 Import dat z NVD

V kapitole 3.3.2 popsaná databáze NVD je zdrojem data pro navrženou aplikaci. Její data jsou poskytována ve dvou formátech XML a JSON, proto bylo nutné zvolit jeden z formátů a tento následně zpracovat pro potřeby aplikace. Jako vhodnější s ohledem na vlastní zkušenosti a čitelnost exportu jsem zvolil formát XML. Ve zdrojových souborech jsou ve velké míře využity tzv. jmenné prostory (namespaces), které umožňují jednoznačnou identifikaci elementů v rámci celého světa. [5]

Import dat z NVD ve formátu XML bylo nutné rozdělit na dvě části. První částí je import velkého počtu technologií (slovník), ke kterým následně patří jednotlivé zranitelnosti, importované v druhé části. Zatímco technologie jsou uloženy v jednom XML souboru, zranitelnosti jsou uloženy v několika souborech rozdělených podle roku zaznamenání daných zranitelností.

Je důležité zmínit, že položky v importu technologií nereprezentují přímo jednotlivé technologie, ale jejich verze. U tohoto způsobu reprezentace dat nejsou data z NVD vždy konzistentní hlavně z důvodu chybějících verzí a také občasné absence tzv. výchozí verze. Při návrhu databáze je totiž nutné mít na paměti i situace, kdy výstupy z detekce nebudou přesné.

Typicky půjde o situaci, kdy bude úspěšně detekována určitá technologie ale nepodaří se detekovat její verzi. Z důvodu složitosti detekce verzí bude k této situaci docházet často a je proto žádoucí zvolit jak vhodný způsob komunikace tohoto stavu uživateli, tak i vhodně uchovávat tuto informaci v databázi. Pro tento účel je pro každou technologii během importu vložena tzv. výchozí verze, představující technologii jako celek bez informace o jakou konkrétní verzi jde.

Využití prostředí příkazů (command)

Symfony Framework poskytuje implicitně podporu pro vytváření tzv. příkazů pro CLI (Command Line Interface). Tyto příkazy jsou vhodné například pro spouštění pomocí `cronu`. Výhodou je možnost nastavit pomocí připravených metod argumenty nebo přepínače jednotlivých příkazů.

Důležitou vlastností příkazů je také jejich přenastavení konfigurace PHP, kdy složité příkazy jejichž provedení zabírá větší množství času, kvůli nastavení `max_execution_time` nemusí doběhnout.

Transformace dat o technologiích

V exportu z obsahujícím data o technologiích jsme schopni získat velkou většinu technologií, které chceme detekovat. Aby však bylo možné tyto informace uložit do připravených tabulek databáze, je nutné provádět při importu jejich transformaci do vhodné podoby.

```
<cpe-item name="cpe:/a:jquery:jquery:1.6">
  <title xml:lang="en-US">jQuery 1.6</title>
  <cpe-23:cpe23-item name="cpe:2.3:a:jquery:jquery:1.6:*:*:*:*:*:*:*"/>
</cpe-item>
```

Výpis 6.1: Ukázka položky technologie v exportu NVD

Ve výpisu 6.1 vidíme, jak vypadá položka technologie v XML exportu. Dříve bylo zmíněno, že v exportu nejsou položky jednotlivé technologie, ale jejich verze. Bylo proto nutné data zpracovávat tak, aby bylo možné vytvořit vazbu mezi technologiemi a jejich verzemi. Import technologií jsem tedy rozdělil do dvou fází.

V první fázi se při průchodu XML dokumentem vytvoří asociativní pole, kde jsou již jednotlivé verze spojeny s danou technologií a zároveň při tomto průchodu dojde k přípravě jednotlivých technologií a jejich verzí.

Hlavními atributy technologií jsou `vendor` a `product`. Tyto dva atributy jsou na úrovni technologií dostačujícím identifikátorem a jejich spojení jsem tedy využil jako hodnotu primárního klíče tabulky `Technology`. Pro verze jednotlivých technologií jsem naopak využil, standardem daný identifikační řetězec v kompletní podobě. U výpisu 6.1 to tedy je `cpe:2.3:a:jquery:jquery:1.6.1:*:*:*:*:*:*:*`. U verzí technologií kromě autora a názvu dochází i k importu názvu verze (obvykle numerické znaky oddělené tečkami) a případné označení vývojové fáze (např. beta, release candidate, ...).

Zpracování dat o zranitelnostech

Ve druhé části importu dat z NVD dochází ke zpracování a importu jednotlivých softwarových zranitelností. Výhodou importu zranitelností není nutnost transformace nebo úprav vstupních dat.

```
<entry id="CVE-2013-0337">
  <vuln:vulnerable-software-list>
    <vuln:product>cpe:/a:nginx:nginx:1.0.0</vuln:product>
    <vuln:product>cpe:/a:nginx:nginx:1.0.1</vuln:product>
    <vuln:product>cpe:/a:nginx:nginx:1.0.2</vuln:product>
    <vuln:product>cpe:/a:nginx:nginx:1.0.3</vuln:product>
  </vuln:vulnerable-software-list>
  <vuln:cve-id>CVE-2013-0337</vuln:cve-id>
  <vuln:published-datetime>2013-10-26T20:55:03</vuln:published-datetime>
  <vuln:cvss>
    <cvss:base_metrics>
      <cvss:score>7.5</cvss:score>
    </cvss:base_metrics>
  </vuln:cvss>
  <vuln:cwe id="CWE-264"/>
  <vuln:summary>The default configuration of...</vuln:summary>
</entry>
```

Výpis 6.2: Ukázka položky zranitelnosti v exportu NVD

Ve výpisu 6.2 můžeme vidět záznam zranitelnosti v XML exportu, přičemž je z tohoto záznamu do databáze potřebné ukládat pouze některé atributy. Nejdůležitějšími informacemi jsou identifikátor zranitelnosti, ve výpisu jde o `CVE-2013-0337`, a také popis zranitelnosti vyskytující se v elementu `vuln:summary`. Mezi důležité části patří i obsah elementu `vuln:vulnerable-software-list`, který obsahuje standardizované identifikátory všech verzí technologií obsahující danou zranitelnost. Vzhledem k tomu, že je tento identifikátor ve stejné formě uložen jako primární klíč pro verze technologií, je možné pomocí něj zranitelnost a verzi technologie jednoduše spojit.

Kromě těchto informací je také ukládán obsah elementu `cvss:score`, kdy tato hodnota definuje skóre zmíněné v kapitole 3.2.1, a obsah elementu `vuln:published-datetime` označující datum, kdy byl záznam o zranitelnosti publikován.

6.1.2 Doplňující import z GitHub API

I přes velký objem dat, který NVD pomocí svých feedů poskytuje, nebyla data o technologiích kompletní. Zejména šlo o chybějící verze technologií, což souvisí s tím, že nevzniká potřeba uchovávat v databázi NVD každou jednotlivou verzi ve chvíli, kdy nemá ani jednu známou zranitelnost. Důvodem je také absence přímé kooperace mezi správcem dat a vývojáři jednotlivých technologií.

Jako řešení tohoto problému jsem se rozhodl využít službu, kde vývojáři naopak s komunitou přímo spolupracují. Velkým zdrojem dat o technologiích jsou totiž jejich repozitáře ve službě GitHub. GitHub je dnes již standardem pro verzování a sdílení zdrojového kódu, kromě toho ale slouží i k reportingu chyb, zaznamenávání jejich oprav a také k uchovávání informací o verzích jednotlivých technologií.

GitHub API umožňuje přístup k velkému množství informací o každém repozitáři, přičemž nejzajímavější je v kontextu řešení probému s chybějícími verzemi právě *endpoint* Releases[1]. Zde jsou dostupné informace o jednotlivých verzích a tvoří tak vhodný zdroj dat pro doplnění neúplných dat z NVD.

[1] https://developer.github.com/v3/repos/releases/#list-releases-for-a-repository

Transformace dat z GitHub API

Stejně jako u dat z NVD je i u dat GitHub API nutné provést jejich transformaci. Kromě transformace dat je z důvodu využití kompletního identifikátoru ze standardu CVE nutné tento identifikátor uměle vytvořit. Kromě nutnosti vytvoření identifikátoru je komplikací i nekonzistentní pojmenování verzí. U technologie PHP například jména verzí obsahují prefix `php-`, a proto bylo nutné tyto „parazitní“ znaky z názvů verzí odstranit.

6.2 Implementace detekce technologií

Stěžejní částí implementace byla logika pro detekci technologií. Na základě informací z kapitoly 2.3 bylo nutné v této části spojit jednotlivé způsoby detekce, a tyto následně na základě předpisu pro každou technologii provést.

6.2.1 Asynchronní provedení detekce

S ohledem na složitost a časovou náročnost procesu detekce je nutné tento proces provádět z pohledu uživatele asynchronně. Kvůli tomu bylo nutné na straně API implementovat vícekrokovou obsluhu pro správu procesu detekce.

Bylo nutné oddělit vyvoření samotného úkolu detekce, jeho spuštění a následné předávání výstupu z detekce. Každý tento krok z pohledu API představuje samostatný *endpoint*. Požadavek na vytvoření úkolu detekce je dostupný na URL `/api/detection/create`, kdy parametrem tohoto požadavku typu `POST` je adresa stránky pro kterou má být detekce provedena. V odpovědi je klientovi zaslán vygenerovaný unikátní identifikátor dané detekce (token), který je nutný pro další postup v procesu detekce.

Spuštění úkolu detekce je dostupné na URL `/api/detection/start`, kdy je očekáván stejně jako u endpointu pro vyvtoření úkolu detekce požadavek typu `POST`. Parametrem však v tomto případě není URL stránky pro detekci ale unikátní identifikátor již vygenerovaného úkolu detekce.

V průběhu detekce jsou výsledky postupně ukládány do databáze, díky tomu se frontend může v časových intervalech dotazovat na aktuální stav detekce a tento stav zobrazovat. Zároveň tak dostane uživatel informaci o tom, zda detekce probíhá nebo již byla ukončena.

6.2.2 Komunikace s prohlížečem Zombie.js

Jak bylo zmíněno v kapitole 5.4.4, je nutné v backend části aplikace zvolit vhodnou knihovnu pro komunikaci s prohlížečem, ve kterém dojde k vykreslení stránky. Během dokončování implementace jsem aplikaci upravil tak, aby využívala prohlížeč Zombie.js, což obnášelo i nutnou změnu knihovny pro komunikaci s tímto prohlížečem, která byla v aplikaci využita.

Instalace knihovny `minkphp/mink-zombie-driver`[2] byla hlavním problémem tohoto přechodu, kdy nebylo možné knihovnu v aktuální verzi nainstalovat kvůli kolizi závislostí. Byl jsem tedy nucen vytvořit vlastní alternativní větev[3] (tzv. fork) této knihovny a v ní upravit její závislosti tak, aby bylo možné knihovnu do aplikace přidat.

[2] https://github.com/minkphp/MinkZombieDriver
[3] https://github.com/reddybednar/MinkZombieDriver

6.2.3 Normalizace URL adres souborů se zdrojovým kódem

V jazyce HTML je možné definovat externí zdrojové soubory pomocí neúplného zadání URL, přičemž dnešní prohlížeče jsou schopny zpracovat i URL adresy bez uvedených částí schéma a doménové jméno. Pokud tedy v HTML kódu použijeme například v atributu `src` elementu `script` hodnotu `/js/main.bundle.js`, prohlížeč během načtení stránky chybějící části přidá. Je však nutné zmínit, že soubor musí být umístěn na stejné doméně jako stránka, ze které je tento soubor odkazován.

Vzhledem k tomu, že má aplikace pracuje přímo se zdrojovým kódem stránky v HTML, musí být tato normalizace adresy provedena i v ní. Pro provedení této normalizace jsem v implementaci využil funkci jazyka PHP `parse_url`[4].

Tato funkce rozdělí URL na jednotlivé části do asociativního pole a díky tomu je možné pomocí funkce `isset` zjistit, které části URL je nutné doplnit. Ve výpisu 6.3 je ukázka kódu, kterým je výše zmíněná normalizace provedena.

```
$urlParts = parse_url($source);
if (!isset($urlParts['host'])) {
  $urlParts['host'] =
    $session->evaluateScript('return window.location.host');
}
if (!isset($urlParts['scheme'])) {
  $urlParts['scheme'] =
    $session->evaluateScript('return window.location.protocol');
}
```

Výpis 6.3: Kód pro provedení normalizace URL adresy

6.2.4 Stažení zdrojových kódů

V kapitole 4.3 je uvedeno stažení zdrojových kódů jako jeden ze způsobů zpracování webové stránky, přičemž umožňuje vhodné způsoby detekce technologií.

Ke stažení javaskriptového kódu jsem využil selekci všech script elementů, což umožňuje stáhnout nejen externě načítané soubory uvedené v atributu `src`, ale i přímo vložený javaskriptový kód v těle elementů. Každý samostatný soubor nebo kód z elementu je poté uložen do dočasného adresáře. Tento adresář je při ukončení detekce smazán aby nedocházelo ke zbytečnému uchovávání již nepotřebných dat. Kromě javaskriptového kódu je pro účely detekce stažen také HTML kód stránky.

Vzhledem k tomu, že na některých webových stránkách může být značené množství externích javaskriptových souborů, postupné stažení těchto souborů by znamenalo dlouhou časovou prodlevu v detekci. Jako řešení jsem využil funkcionalitu HTTP klienta `GuzzleHttp`[5], díky které je možné definovat pole požadavků na požadované soubory a při odeslání jsou poté tyto požadavky odeslány konkurentním způsobem. Takto jsou poté zpracovány i odpovědi. Tímto řešením bylo dosaženo velké časové úspory.

6.2.5 Předpis pro detekci jednotlivých technologií

Pro technologie, které je možné na webových stránkách detekovat, je nutné specifikovat znaky, pomocí kterých je možné danou technologii rozpoznat. Vzhledem k tomu, že aplikace

[4] https://www.php.net/manual/en/function.parse-url.php
[5] http://docs.guzzlephp.org/en/stable/

využívá několik způsobů detekce, pro každý typ detekce může být definováno určité pravidlo, při jehož splnění aplikace vyhodnotí, že je technologie na stránce využita.

Na základě informací zjištěných v kapitole 2.3 jsem navhnul formát předpisu, ve kterém je možné pro jednotlivé způsoby detekce zmíněná pravidla definovat. Předpis je poté součástí zdrojového kódu aplikace.

```
"React" => [
  "CODE" => [
    "js" => [
      "reactDOM",
      "react-dom",
      "\/\*\* @license React"
    ],
  ],
  "VERSION" => [
    "CODE" => [
      "js" => [
        "\/\*\* @license React v(\d+\.?\d+\.?\d+)",
      ],
    ],
  ],
]
```

Výpis 6.4: Předpis pravidel pro technologii React.js

Ve výpisu 6.4 vidíme předpis pro technologii React.js. V části `CODE`, je možné specifikovat zda se pravidlo týká javaskriptového nebo HTML kódu, jednotlivé položky poté už představují jednotlivá pravidla. Pravidla je možné zadávat jako regulární výrazy i jako klasické řetězce.

V části `VERSION` nastavujeme pravidla pro extrakci verze dané technologie. Verzi lze extrahovat nejen ze zdrojového kódu, ale i z hlaviček HTTP odpovědi nebo window objektu dané stránky. V případě, že extrahujeme verzi z kódu, je poté opět nutné specifikovat typ kódu a následně definovat pravidlo pro extrakci verze.

Takto specifikovaný formát umožňuje další rozvoj aplikace vzledem k množství detekovaných technologií, přičemž pro vytvoření tohoto předpisu není nutná znalost kódu aplikace.

6.3 Implementace uživatelského rozhraní

Pro implementaci požadovaných funkcionalit uživatelského rozhraní, definovaných v kapitole 5.6, jsou některých případech použity zajímavá řešení a přístupy. V této kapitole jsou některé tyto postupy popsány a také jsou zde obsaženy obrázky uživatelského rozhraní, pro lepší názornost jednotlivých částí uživatelského rozhraní aplikace.

6.3.1 Komunikace komponent s JSON API

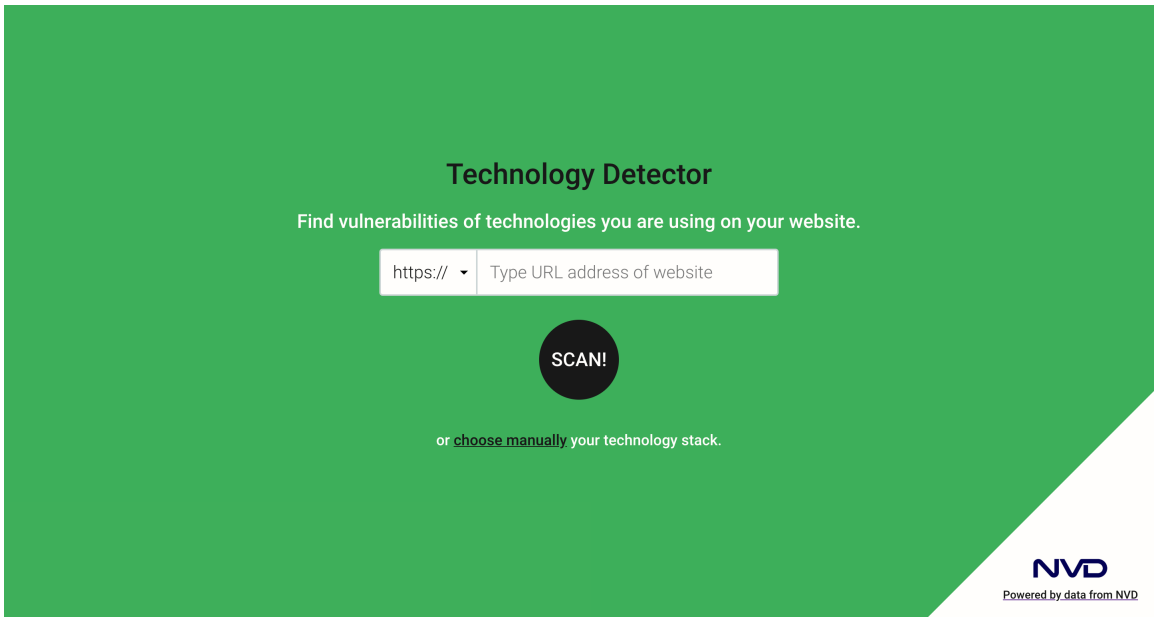
Pro komunikaci mezi komponentami a backend částí aplikace je v aplikaci použito JSON API popsané v kapitole 5.1.1. Komponenty z API získají zvůj výchozí stav a také pomocí API ukládají případné změny vykonané uživatelem.

Komunikace probíhá pomocí klasických HTTP požadavků. K tomuto účelu jsem se v aplikaci rozhodl použít knihovnu `axios`[6]. Poskytuje jednoduché rozhraní pro zasílání požadavků a následnou práci s odpovědí nebo pro případné zachytávání chyb při provádění požadavků. Při implementaci bylo nutné uvědomit si, že veškerá komunikace mezi API a komponentami probíhá asynchronně, je proto nutné využívat tzv. *callback* funkce. Takové funkce jsou provedeny až po dokončení událostí, na kterou jsou navázány.

6.3.2 Zpracování adresy při vložení (copy-paste)

Do pole pro zadání adresy na hlavní stránce aplikace bude velmi často uživatel adresu vkládat pomocí klávesové zkratky. Proto jsem se rozhodl v aplikaci implementovat funkcionalitu, která při vyvolání události vložení (paste) zjistí, zda vkládaná adresa obsahuje protokol HTTP nebo HTTPS.

Na základě zjištěného protokolu je poté automaticky oddělena tato část (`http://` nebo `https://`) od adresy a nastavena právě v elementu pro výběr protokolu. Do textového pole pro adresu je poté adresa vložena již bez protokolu aby byl zachován formát, kdy je protokol zvolen v elementu `select` a adresa je specifikována v elementu `input`. Při odeslání formuláře poté pouze dojde ke konkatenaci hodnot z těchto dvou elementů. Elementy popsané výše je možné vidět na obrázku 6.1.



Obrázek 6.1: Hlavní stránka aplikace

6.3.3 Kontrola existence stránky

Po odeslání formuláře je nutné zabránit situaci, kdy je spuštěna detekce pro neexistující webovou stránku. Kontrolu existence stránky jsem se proto rozhodl implementovat jako součást vytvoření úkolu detekce. Při zaslání požadavku na adresu `/api/detection/create` je z backend části aplikace pomocí knihovny `GuzzleHttp` odeslán požadavek na uživatelem vloženou adresu.

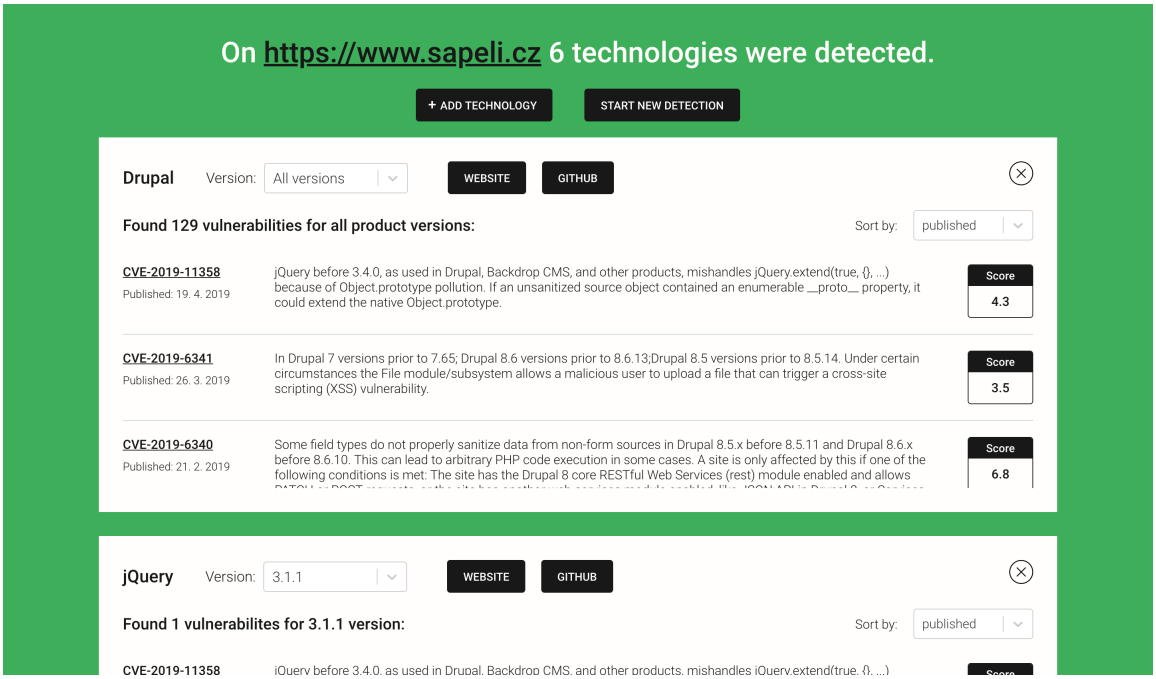
[6] https://github.com/axios/axios

V případě, že požadavek není naplněn, odešle API v odpovědi informaci o chybě a uživateli tak je zobrazeno upozornění, že pro webovou stránku na zadané adrese není možné detekci provést.

6.3.4 Výsledek detekce

Po stisknutí tlačítka „Scan!“, čímž dojde k odeslání formuláře, přichází na řadu asynchronní provedení detekce vysvětlné v kapitole 6.2.1. Díky tomuto přístupu dojde k přesměrování na stránku s výsledkem a uživateli je zobrazena informace, že probíhá detekce. Nedojde tak k situaci, že se uživateli výsledek bez jakékoli informace dlouho načítá. V takové situaci drtivá většina uživatelů aplikace opouští.

Pro aktualizaci stavu detekce probíhají za využití funkce `setInterval` v intervalu dvou vteřin požadavky na API a na základě odpovědi dojde k úpravě stránky s výsledkem. Po dokončení detekce je odstraněna informace o probíhající detekci a výsledek tak lze považovat za finální. U finálního výsledku je zobrazen počet detekovaných technologií a je zde také pro účely sdílení zobrazena adresa stránky, pro kterou byla detekce provedena, jak je vidět na obrázku 6.2.



Obrázek 6.2: Stránka s výsledkem detekce obsahující seznam detekovaných technologií

6.3.5 Modální okno pro výběr technologie

Modální okno pro výběr technologie má dva stavy. Ve výchozím stavu jsou vypsány abecedně seřazené technologie, přičemž při scrollování dochází k jejich postupnému donačítání. Druhý stav okna je v režimu vyhledávání. Po odeslání vyhledávacího formuláře v horní části okna je kompletní výpis technologií nahrazen technologiemi, které odpovídají zadanému dotazu.

Po nalezení požadované technologie může uživatel jednoduše jedním kliknutím technologii přidat. Ve chvíli, kdy uživatel dokončí přidávání technologií, opustí modální okno

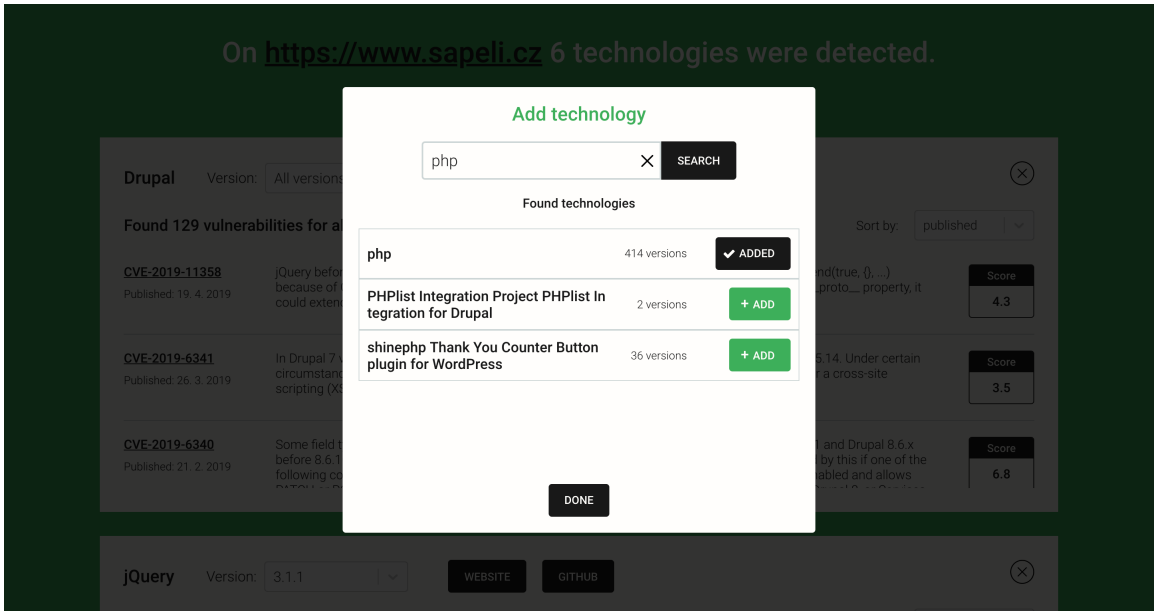
kliknutím na tlačítko v dolní části modálního okna nebo může kliknout na tmavou část mimo toto okno. Modální okno je zobrazeno na obrázku 6.3.

Pokud chce následně uživatel u přidaných technologií specifikovat verzi, může tak učinit v už přidané komponentě technologie. Toto řešení je z uživatelského pohledu přívětivější a odstraňuje tak nutnost mít přidání technologie ve formě více kroků.

Zobrazení informace o přidání technologie

Zajímavým řešením je zobrazení informace, že je technologie již ve výsledku přidána. Vzhledem k tomu, že je modálním oknem překryt seznam přidaných technologií, uživatel neztratí přehled o tom, které technologie již přidal nebo jsou ve výsledku obsaženy. Výhodou použitého řešení je také možnost rychlého přidání více technologií najednou.

Aby bylo možné tuto funkcionalitu implementovat, je seznam již obsažených technologií předáván z komponenty `Result` pomocí `props` jednotlivých komponent až do komponenty `AddTechnologyModal`. V této komponentě je poté implementována metoda `isTechnologyPresent`, která je volána při renderování komponent `TechnologySearchResult`, zobrazených jak ve výsledcích vyhledávání, tak i v seznamu nejpoužívanějších technologií.



Obrázek 6.3: Modální okno pro vyhledání a následné přidání technologie do seznamu technologií

6.3.6 Výpis zranitelností

V rámci komponenty `Technology` je zobrazen výpis zranitelností. V jeho horní části je na levé straně zobrazen textový popis oznamující počet zranitelností nalezených pro danou verzi a na pravé straně výběr atributu, podle kterého mají být zranitelnosti seřazeny. Vzhledem ke kontextu se jako nejvhodnější atributy pro řazení jeví datum publikace zranitelnosti a její závažnost, tedy skóre.

Vzhledem k tomu, že výpis zranitelností může být v některých situacích obsáhlý, zvolil jsem řešení, kdy je výška výpisu omezena a je v něm umožněno vertikální scrollování.

Kapitola 7

Porovnání s existujícími nástroji

Pro vyhodnocení úspěšnosti detekce jsem se rozhodl zvolit porovnání s existujícím nástrojem Wappalyzer, který je popsán v kapitole 2.2.3. Ačkoliv Wappalyzer detekuje mj. velké množství technologií, které s ohledem na existenci zranitelností nejsou pro mou práci zajímavé, provedl jsem pro zvolené webové stránky detekci pomocí obou aplikací.

7.1 Volba webových stránek a provedení detekce

Pro porovnání jsem zvolil webové stránky využívající široké spektrum technologií tak, aby bylo prověřeno správné fungování obou nástrojů. Zvolené webové stránky využívají jak javaskriptové knihovny, tak i serverové technologie nebo redakční systémy.

První webovou stránkou pro detekci je hlavní stránka webu Fakulty informačních technologií VUT v Brně. Výsledky detekce byly kromě knihovny jQuery UI stejné a i při zběžné manuální kontrole reálného využití těchto technologií výsledky odpovídají. Přehled je znázorněn v tabulce 7.1.

Tabulka 7.1: Výsledky detekce pro adresu https://www.fit.vutbr.cz

Název technologie	Aplikace Wappalyzer	Implementovaná aplikace	Reálné použití
Apache	Ano	Ano	Ano
PHP	Ano	Ano	Ano
jQuery	Ano	Ano	Ano
jQuery UI	Ano	Ne	Ano

Vzhledem k otestování detekce modernějších technologií jsem jako další stránku zvolil hlavní stránku webu pro vývojáře služby Spotify. U této webové stránky se výsledky detekce obou aplikací velmi lišily a bylo tedy nutné provést manuální kontrolu. Aplikace Wappalyzer detekovala technologii AngularJS, zatímco mnou implementovaná aplikace detekovala React.js. V tomto případě jde o zcela protichůdné informace a po manuální kontrole bylo zjištěno, že stránka využívá knihovnu React.js. Kromě správné technologie také mnou implementovaná aplikace detekovala i verzi, ve které byla knihovna React.js použita, tedy verzi 16.7.0. Přehled je znázorněn v tabulce 7.2.

Další webovou stránkou vhodnou k detekci je webová stránka společnosti Sapeli. Vzhledem k tomu, že mám díky pracovním zkušenostem přehled jaké technologie jsou na stránce využity, zvolil jsem tuto stránku právě kvůli vyššímu počtu detekovatelných technologií. V tomto porovnání obě aplikace vyhodnotily technologie správně a nedošlo tedy k žád-

Tabulka 7.2: Výsledky detekce pro adresu https://developer.spotify.com/

Název technologie	Aplikace Wappalyzer	Implementovaná aplikace	Reálné použití
AngularJS	Ano	Ne	Ne
jQuery	Ne	Ano	Ano
React.js	Ne	Ano + verze	Ano

nému rozporu ve výsledcích. Vyjímkou je pouze technologie MySQL, nicméně zde moje aplikace vychází z předpokladu, že technologie Drupal využívá databázi MySQL, což u některých vyjímek nemusí platit. Přehled detekovaných technologií je znázorněn v tabulce 7.3.

Tabulka 7.3: Výsledky detekce pro adresu https://www.sapeli.cz/

Název technologie	Aplikace Wappalyzer	Implementovaná aplikace	Reálné použití
Drupal	Ano	Ano	Ano
jQuery	Ano	Ano + verze	Ano
MySQL	Ne	Ano	Ano
Nginx	Ano	Ano	Ano
PHP	Ano	Ano	Ano
Varnish	Ano	Ano	Ano

7.2 Vyhodnocení výsledků

Dle výše provedených porovnání hodnotím z pohledu mnou implementované aplikace výsledky pozitivně. V širokém spektru technologií byla aplikace schopna většinu z nich detekovat, a dokonce u stránky pro vývojáře služby Spotify správně detekovat knihovnu React.js, zatímco aplikace Wappalyzer detekovala knihovnu AngularJS.

Jako další pozitivum je třeba uvést i detekci verze u dvou technologií, konkrétně u knihoven jQuery a React.js. Jsem přesvědčen, že při dalším rozvoji aplikace a pravidel pro detekci by byla úspešnost detekce verzí ještě vyšší.

Kapitola 8

Závěr

Cílem mé práce bylo vytvoření služby pro detekci technologií použitých na webových stránkách a výpis jejich zranitelností. Tuto službu jsem se rozhodl implementovat jako webovou aplikaci založenou na datech z externích zdrojů.

Na začátku práce byl proveden průzkum možností pro detekci technologií, ze kterých byly následně zvoleny ty nejvhodnější. Dále bylo nutné prozkoumat a vyhodnotit, které nástroje bude nutné využít pro zpracování webových stránek, aby pro ně bylo pomocí těchto nástrojů možné detekci provést.

Vzhledem k požadavkům na funkcionality služby bylo nutné seznámit se s existujícími zdroji dat v oblasti technologií a jejich zranitelností. Jako zdroj dat pro implementovanou službu byla zvolena databáze NVD. Po výběru zdroje dat bylo nutné zvolit i vhodný způsob jejich zpracování a následného uchovávání. Ačkoliv je formát dat z NVD standardizovaný a dobře zdokumentovaný, transformace těchto dat se ukázala být nejnáročnějším úkolem v rámci celé implementace.

Na základě průzkumu byl vytvořen návrh aplikace, implementující požadované funkcionality. V rámci návrhu bylo nutné zvolit vhodné implementační technologie a koncepty. Pro implementaci navržené aplikace jsem zvolil moderní nástroje s širokou komunitou a velkým potenciálem i z pohledu možného využití mnou získaných zkušeností v praxi.

Součástí práce byl také návrh a implementace jednoduchého uživatelského rozhraní. Vzhledem ke komplexnosti aplikace se mi bohužel nepovedlo zaměřit se na některé části rozhraní detailněji, proto vidím právě uživatelské rozhraní jako oblast, na kterou by se dalo při dalším rozvoji aplikace zaměřit nejvíce. Při implementaci služeb pro detekci technologií jsem narazil na několik problémů, a to zejména při instalaci a nastavení aplikace na produkční server. Všechny problémy se mi však povedlo vyřešit, i když jejich řešení zabralo více času, než jsem předpokládal.

V implementované aplikaci je aktuálně připraven předpis pro detekci 21 technologií, které obsahují celkem 5792 verzí. U deseti technologií je také připraven předpis pro detekci verze, přičemž detekce verze je jednou z nejzajímavějších funkcionalit aplikace. Databáze také obsahuje 1339 zranitelností technologií, které poté může uživatel u jednotlivých technologií procházet. Dále je v aplikaci možno manuálně přidat dalších 591 technologií, které převážně doplňují detekovatelné technologie. Pro ukázku je aplikace předpřipravena na adrese https://bp.radekbednar.cz.

V závěru mé práce byla implementovaná aplikace porovnána s již existujícím nástrojem Wappalyzer. Ačkoliv Wappalyzer detekuje i technologie, které v kontextu zranitelností nejsou zajímavé, vyšla mnou implementovaná aplikace z porovnání s dobrými výsledky.

Během práce jsem měl možnost nově pracovat s technologií React.js a vyzkoušet si způsob propojení backend části a frontend části aplikace pomocí JSON API, což z pohledu nových zkušeností hodnotím velmi pozitivně. Také jsem měl možnost prozkoumat jakým způsobem jsou zpracovávány a ohodnocovány softwarové zranitelnosti, což byla jedna z nejzajímavějších oblastí studijní části mé práce.

Mezi hlavní problémy a chyby řadím zejména objevení nástroje Wappalyzer až ve fázi implementace aplikace. Má aplikace oproti nástroji Wappalyzer ztrácela zejména v rychlosti detekce. Na konci práce se mi však podařilo aplikaci upravit tak, aby využívala bezhlavičkový prohlížeč Zombie.js, čímž se mi podařilo zkrátit dobu detekce na polovinu oproti původní implementaci.

Aplikace má velký potenciál pro další rozvoj, ať už jde o rozšíření počtu detekovaných technologií, tak i o optimalizaci procesu detekce. Při implementaci jsem se snažil aplikaci na další rozvoj připravit, což mě vedlo k návrhu a použití formátu předpisu pro detekci technologií, který může doplňovat i vývojář bez hlubší znalosti implementace. Další zajímavou oblastí rozvoje může být integrace umělé inteligence do procesu detekce, kdy by bylo možné automatické generování předpisů pro detekci nebo jejich aktualizace v závislosti na změnách v kódu detekovaných webových stránek.

Literatura

[1] C. Zakas, N.: *High Performance JavaScript - Build Faster Web Application Interfaces.* O'Reilly, 2010, ISBN 978-0-596-80279-0.

[2] Facebook: *facebook.github.io – React Native - Build native mobile apps using JavaScript and React.* [Online; navštíveno 17. 12. 2018].
URL https://facebook.github.io/react-native/

[3] Foundation, N.: *nodejs.org – About | Node.js.* [Online; navštíveno 4. 2. 2019].
URL https://nodejs.org/en/about/

[4] Hund, P.: *https://medium.freecodecamp.org – SEO vs. React: Web Crawlers are Smarter Than You Think.* [Online; navštíveno 10. 5. 2019].
URL https://medium.freecodecamp.org/seo-vs-react-is-it-neccessary-to-render-react-pages-in-the-backend-74ce5015c0c9

[5] Kosek, J.: *PHP a XML.* Grada Publishing a.s., Praha, 2009, ISBN 978-80-247-1116-4.

[6] Michel, J.: *github.com – ChromeLibraryDetector - Google Chrome extension for JavaScript libraries detection.* [Online; navštíveno 17. 12. 2018].
URL https://github.com/johnmichel/Library-Detector-for-Chrome/blob/master/library/libraries.js

[7] Mishra, R.: *hackernoon.com – Virtual DOM in ReactJS.* [Online; navštíveno 12. 4. 2019].
URL https://hackernoon.com/virtual-dom-in-reactjs-43a3fdb1d130

[8] MITRE: *cve.mitre.org – CVE - A list of common software vulnerabilities.* [Online; navštíveno 17. 12. 2018].
URL https://cve.mitre.org/index.html

[9] MITRE: *cve.mitre.org – CVE - CVE and NVD Relationship.* [Online; navštíveno 17. 12. 2018].
URL https://cve.mitre.org/about/cve_and_nvd_relationship.html

[10] MITRE: *cwe.mitre.org – CWE - A community-developed list of software weakness types.* [Online; navštíveno 17. 12. 2018].
URL https://cwe.mitre.org/index.html

[11] MITRE: *cwe.mitre.org – CWE - Weaknesses in OWASP Top Ten.* [Online; navštíveno 17. 12. 2018].
URL https://cwe.mitre.org/data/definitions/1026.html

[12] Pivetta, M.: *ocramius.github.io/blog – Doctrine ORM Hydration Performance Optimization*. [Online; navštíveno 5. 5. 2019].
URL
https://ocramius.github.io/blog/doctrine-orm-optimization-hydration/

[13] Schroeder, K.: *eschrade.com – Performance of Apache 2.4 with the event MPM compared to nginx*. [Online; navštíveno 3. 5. 2019].
URL https://www.eschrade.com/page/performance-of-apache-2-4-with-the-event-mpm-compared-to-nginx/

[14] W3C: *w3.org – DOM4*. [Online; navštíveno 31. 12. 2018].
URL https://www.w3.org/TR/domcore/

[15] W3C: *w3.org – WebDriver*. [Online; navštíveno 28. 12. 2018].
URL https://www.w3.org/TR/webdriver1/

[16] W3C: *w3schools.org – XPath*. [Online; navštíveno 6. 1. 2019].
URL https://www.w3schools.com/xml/xpath_intro.asp

Příloha A

Obsah přiloženého paměťového média

- `/app-src/` - zdrojový kód implementované aplikace
- `/tex-src/` - zdrojový kód textu práce
- `/pdf/` - text práce ve formátu PDF

Příloha B

Zprovoznění aplikace v lokálním prostředí

Pro zprovoznění aplikace na lokálním zařízení, je třeba provést následující kroky v uvedeném pořadí.

B.1 Instalované programy

Na zařízení musí být nainstalovány tyto programy:

- `PHP` ve verzi >= 7.1.3
- `Node` ve verzi >= 10
- Balíčkovací systém `npm` ve verzi >= 6.0
- `MySQL` ve verzi >= 5.6
- Globálně nainstalovaný správce závislostí `Composer`

B.2 Instalace aplikace

Po přepnutí do kořenového adresáře projektu (adresář obsahující soubory `composer.json` a `package.json`) je třeba spustit následující příkazy:

- `composer install`
- `npm install`

Po doběhnutí instalace závislostí je třeba vytvořit v kořenovém adresáři projektu soubor s názvem `.env.dev.local`. Do tohoto souboru je poté nutné vložit tento řádek:

- `DATABASE_URL=mysql://DB_USER:DB_PASSWORD@127.0.0.1:3306/DB_NAME`

Parametry `DB_USER`, `DB_PASSWORD` a `DB_NAME` je samozřejmě nutné nahradit hodnotami odpovídajícími lokálnímu nastavení.

Po vytvoření souboru `.env.dev.local` je nutné (v případě, že tak již nebylo učiněno) vytvořit databázi. Databázi lze vytvořit příkazem:

- `bin/console doctrine:database:create`

Dále je nutné v databázi vytvořit schéma. Pro vytvoření schématu je nutné spustit přikaz:

- `bin/console doctrine:schema:create`

B.3 Import technologií a zranitelností

Jako první krok po úspěšné instalaci musí být databáze naplněna daty z databáze NVD. V první části je nutné naimportovat technologie a jejich verze:

- `bin/console import-dictionary`
- `bin/console import-missing-technologies`
- `bin/console import-github GITHUB_USERNAME GITHUB_PASSWORD`

Parametry GITHUB_USERNAME GITHUB_PASSWORD je nutné nahradit přihlašovacími údaji do služby GitHub. Přihlášení je nutné kvůli vyššímu limitu počtu požadavků.

Po importu technologií je již možné importovat zranitelnosti:

- `bin/console import-vulnerabilities 2019`

Číslo na konci příkazu definuje rok, pro který se mají zranitelnosti importovat. Databáze NVD obsahuje zranitelnosti od roku 1999 až do aktuálního. Pro úplný import zranitelností je tedy nutné spustit přikaz pro roky od 1999 do aktuálního.

Pro pravidelné produkční aktualizace je poté možné v cronu nastavit pouze import posledních několika let, kdy dochází ke změnám.

B.4 Spuštění aplikace

Pro spuštění aplikace na lokálním zařízení lze využít PHP server, spustitelný příkazem:

- `bin/console server:run`

Po spuštění příkazu bude aplikace dostupná na adrese `http://localhost:8000`.

B.5 Úpravy frontend části aplikace

Pro úpravy frontend části aplikace (HTML, CSS, JS) je možné spustit tzv. watcher, který provede automatickou kompilaci při každé změně zdrojového kódu. Tento watcher lze spustit příkazem:

- `npm run dev-server`

Po dokončení úprav je nutno zkompilovat zdrojové kódy do produkční verze. Toho lze docílit pomocí přikazu:

- `npm run build`

B.6 Úpravy předpisu detekce

Předpis pro detekci jednotlivých technologí je uveden ve třídách:

- `src/Service/TechnologyStack/CMS.php`
- `src/Service/TechnologyStack/JSLibraries.php`
- `src/Service/TechnologyStack/Server.php`