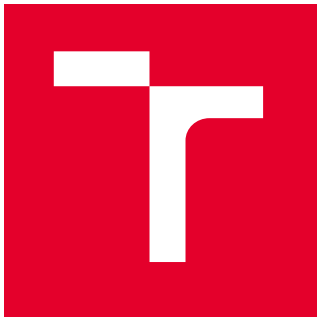
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

DIPLOMOVÁ PRÁCE

Brno, 2022 Bc. Dominik Richter



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ

DEPARTMENT OF TELECOMMUNICATIONS

TVORBA WEBOVÉ APLIKACE V PHP PRO PENETRAČNÍ TESTOVÁNÍ SYSTÉMU PRESTASHOP

PHP WEB APPLICATION FOR PENETRATION TESTING OF THE PRESTASHOP SYSTEM

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

AUTOR PRÁCE — **Bc. Dominik Richter**

AUTHOR

VEDOUCÍ PRÁCE — **Ing. Tomáš Slunský**

SUPERVISOR

BRNO 2022

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní program **Informační bezpečnost**

Ústav telekomunikací

Student: Bc. Dominik Richter ***ID:*** 201350

Ročník: 2 ***Akademický rok:*** 2021/22

NÁZEV TÉMATU:

Tvorba webové aplikace v PHP pro penetrační testování systému PrestaShop

POKYNY PRO VYPRACOVÁNÍ:

Seznamte se s platformou PrestaShop a aktuálním vývojem v oblasti penetračního testování. Dále se seznamte s PHP frameworkem Laravel a REST API. Na základě zjištěných poznatků navrhněte sadu penetračních testů pro platformu PrestaShop včetně architektury webové aplikace. Navržené řešení implementujte formou uživatelsky přívětivé webové aplikace s možností testování jednotlivých útoků. Dosažené výsledky zhodnoťte a vhodně prezentujte.

DOPORUČENÁ LITERATURA:

[1] Laravel - The PHP Framework For Web Artisans [online]. Arkansas: Taylor Otwell, 2021 [cit. 2021-9-14]. Dostupné z: https://laravel.com/

[2] Create and build your online store with PrestaShop [online]. Paris, France: PrestaShop, 2007 [cit. 2021-9-14]. Dostupné z: https://www.prestashop.com

Termín zadání: 7.2.2022 ***Termín odevzdání:*** 24.5.2022

Vedoucí práce: Ing. Tomáš Slunský

Konzultant: Ing. Michal Macek

doc. Ing. Jan Hajný, Ph.D.
předseda rady studijního programu

ABSTRAKT

Diplomová práce je zaměřena na vývoj aplikace v programovacím jazyce PHP pro penetrační testování webových aplikací využívajících systém PrestaShop. Podobně jako PrestaShop jsou i další platformy zprostředkovávající implementaci internetových obchodů velmi exponovaným kontaktním bodem se zákazníky. Proto jsou také cílem mnoha kybernetických útoků, proti kterým je potřeba je chránit. V teoretické části práce jsou čtenáři představeny technologie PHP, MySQL nebo framework Laravel a architektura webových aplikací MVC včetně REST API. Dále je detailně popsán systém PrestaShop a dělení a metodologie penetračního testování. V praktické části práce je představeno vývojové a testovací prostředí a následně popsána webová aplikace PrestaCure s implementovanou sadou penetračních testů. Výsledky práce ukazují plnou funkčnost a využitelnost implementované aplikace v praxi, i s ohledem na jednoduchost a modularitu přidávání dalších penetračních testů.

KLÍČOVÁ SLOVA

PrestaShop, PHP, MySQL, MVC, Laravel, penetrační testování, webová aplikace

ABSTRACT

This diploma thesis is focused on the development of an application in PHP programming language for penetration testing of web other applications using PrestaShop system. Similar to PrestaShop, other platforms mediating the implementation of online stores are a very exposed point of contact with customers. Therefore, they are also the target of many cyber-attacks against which they need to be protected. In the theoretical part of the thesis, the reader is introduced to PHP, MySQL or Laravel framework technologies and MVC web application architecture including REST API. Furthermore, the PrestaShop system and the penetration testing methodology are described in detail. In the practical part of the thesis, the development and testing environment is introduced and the PrestaCure web application with implemented penetration testing suite is described. The results of the thesis show the full functionality and usability of the implemented application in practice also with respect to the simplicity and modularity of adding additional penetration tests.

KEYWORDS

PrestaShop, PHP, MySQL, MVC, Laravel, penetration testing, web application

Vysázeno pomocí balíčku `thesis` verze 4.07; `http://latex.feec.vutbr.cz`

RICHTER, Dominik. *Tvorba webové aplikace v PHP pro penetrační testování systému PrestaShop*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2021, 63 s. Diplomová práce. Vedoucí práce: Ing. Tomáš Slunský

Prohlášení autora o původnosti díla

Jméno a příjmení autora:	Bc. Dominik Richter
VUT ID autora:	201350
Typ práce:	Diplomová práce
Akademický rok:	2021/22
Téma závěrečné práce:	Tvorba webové aplikace v PHP pro penetrační testování systému PrestaShop

Prohlašuji, že svou závěrečnou práci jsem vypracoval samostatně pod vedením vedoucí/ho závěrečné práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené závěrečné práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora*

*Autor podepisuje pouze v tištěné verzi.

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu diplomové práce panu Ing. Tomáši Slunskému za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci. Dále bych chtěl poděkovat své přítelkyni a rodině za podporu při studiu a tvorbě práce.

Obsah

Seznam obrázků

Seznam výpisů

Úvod

Diplomová práce se zabývá bezpečností webových aplikací využívajících open-source platformu PrestaShop. Tento systém patří mezi nejpoužívanější e-commerce řešení spravující internetové obchody (e-shopy) v celosvětovém měřítku [1]. Jelikož jsou webové aplikace v podobě e-shopů stále častěji voleny jako hlavní kontaktní bod s uživateli a zákazníky, soudě podle jejich rapidnímu početnímu růstu, měl by být kladen stále větší důraz na jejich bezpečnost [2]. Tento aspekt se ale v dnešním světě často opomíjí.

Většina těchto platforem se zakládá na modulární architektuře, kdy se k hlavnímu jádru systému přidávají různé další součásti (moduly), a je tedy poměrně náročné udržet bezpečnost systému jako celku. Zejména v případech, kdy jsou tyto moduly vyvíjeny dalšími subjekty. Není proto v silách autorů platformy detailněji testovat všechny dostupné moduly a zjišťovat jejich dopad na bezpečnost systému. Navíc na poli e-commerce, a stejně tak i obecně na poli informačních technologií, je stále kladen důraz spíše na aspekty výkonu a rychlosti než na bezpečnost, která stále působí jako poměrně nová, i když se jí v posledních letech připisuje větší a větší důležitost. Je proto důležité se na tuto oblast zaměřit, zejména kvůli celkově vzrůstajícímu počtu kybernetických útoků nejen na webové aplikace [3].

Cílem této diplomové práce je analýza aktuální situace v oblasti penetračního testování platformy PrestaShop. Ze zjištěných poznatků bude následně možné efektivně navrhnout a realizovat webovou aplikaci, která bude pomocí sady penetračních testů schopna otestovat webové aplikace využívající platformu PrestaShop. Její uživatelé by tak získali možnost automatizovaně otestovat vlastní webovou aplikaci a zjistit, jestli neobsahuje kritické či jiné bezpečnostní chyby, které by následně mohli opravit.

První kapitola práce je zaměřena na teorii webových aplikací. Jsou rozebrány technologie využité pro vývoj aplikace - programovací jazyk PHP, databázový systém MySQL a framework Laravel. Dále jsou vysvětleny architektury webových aplikací Model-View-Controller (MVC) a Representational state transfer (REST). Jsou zmíněny také návrhové vzory webových aplikací.

Druhá kapitola se zaměřuje na open-source systém PrestaShop a popisuje jeho využití, architekturu, bezpečnost, rozebírá nejčastěji zneužívané zranitelnosti a současnou situaci.

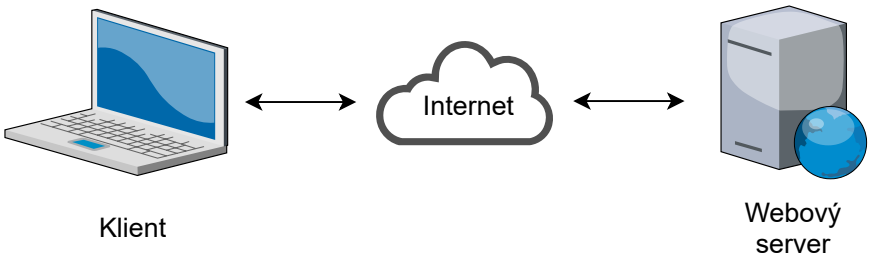
Třetí kapitola teoreticky rozebírá problematiku penetračního testování. Je provedeno rozdělení penetračních testů podle několika kritérií a popsána metodologie testování včetně známého projektu OWASP.

Čtvrtá kapitola se věnuje návrhu vývojového prostředí a jeho součástem. Ve druhé části potom popisuje vytvořený virtuální stroj pro jednoduché možnost otestování vytvořené webové aplikace bez nutnosti instalace a získávání testovaných subjektů.

Poslední kapitola se zabývá popisem implementace webové aplikace PrestaCure a jejích vnitřních procesů. Je rozebrána použitá architektura, vysvětlena struktura databáze a jsou také popsány implementované penetrační testy včetně dokumentace implementace nových penetračních testů. Rovněž je poskytnuta dokumentace k REST API, které je možné použít jako druhý kanál poskytující stejné možnosti jako webová aplikace. Na závěr je zmíněn i samotný postup instalace.

1 Webová aplikace

Webová aplikace je počítačový program spuštěný na vzdáleném webovém serveru, který uživatelům internetu poskytuje požadovaný obsah prostřednictvím komunikace s webovým prohlížečem – klientem. Jejich vzájemná komunikace, znázorněná na obrázku 1.1, probíhá prostřednictvím protokolu Hypertext Transfer Protocol (dále jen HTTP), tedy protokolu aplikační vrstvy modelu ISO/OSI [4].



Obr. 1.1: Komunikace mezi webovým serverem a klientem

Webová aplikace se skládá ze dvou komponent – tzv. front-endu a back-endu. Front-end je ta část aplikace, se kterou uživatel přímo interaguje skrze webový prohlížeč, zobrazující zpracovaný kód webové stránky. Kód formuje tzv. hypertextový dokument posílaný zmíněným protokolem HTTP/HTTPS (zabezpečená verze), který je vytvořen pomocí značkovacího jazyka Hypertext Markup Language (dále jen HTML). Jednotlivé prvky jsou stylizovány prostřednictvím kaskádových stylů Cascading Style Sheets a případný JavaScript dodá rozšířenou funkcionalitu – kód se vykonává až v prohlížeči klienta [4].

Funkční jádro webové aplikace potom tvoří tzv. back-end. Dochází zde k provádění veškeré logiky, ke komunikaci s databází, a také k dynamickému vytváření obsahu za použití programovacích jazyků, jako například Python, PHP, JavaScript, Java nebo C#. Po provedení všech operací webová aplikace předá vytvořený hypertextový dokument webovému serveru, který obsah odesílá klientovi [4].

Server buď v případě statických stránek rovnou odesílá klientovi již připravený hypertextový dokument (např. i uložený v mezipaměti), nebo jej nejdříve dynamicky vytváří s ohledem na klientem vykonanou akci (např. na informace poskytnuté prostřednictvím formuláře) nebo na další procesy uvnitř aplikace (dat z databáze nebo jiné logiky) [4].

V následujících kapitolách budou probrány technologie (1.1.1, 1.1.2) a architektura webových aplikací (1.2.2, 1.2.1), které jsou využity v praktické části práce.

1.1 Využité technologie

Důležitým aspektem při vývoji webové aplikace je výběr technologie. Postupným vývojem vzniklo množství přístupů, které je možné zvolit při jejich implementaci. Vývojář si tak může vybrat, ve kterém programovacím jazyce bude webovou aplikaci vytvářet, případně jaký přidružený framework využije.

1.1.1 PHP a MySQL

Pro praktickou implementaci byl zvolen široce používaný open-source skriptovací objektově orientovaný jazyk pro tvorbu webových aplikací PHP (Hypertext Preprocessor) – podle zdroje [5] jej na svém back-endu využívá 78,4% webů. Byl vytvořen v roce 1994 Rasmusem Lerdorfem, který původně vycházel z jazyka C. PHP je šířeno pod open-source licencí PHP License v3.01 [6].

Jazyk PHP lze používat ve všech hlavních operačních systémech. Mezi jeho největší výhodu patří integrace do jazyka HTML, která tak činí kód velice přehledným, a zároveň jednoduchým i pro začátečníky. V této práci je využita jeho nejnovější verze PHP 8.0. PHP ve svém jádru rovněž obsahuje různé open-source knihovny pro přístup k serverům FTP (File Transfer Protocol) a mnoha databázovým serverům, včetně PostgreSQL, SQLite a MySQL [6].

Databáze MySQL tvoří s jazykem PHP populární dvojici, a proto je v využita v praktické implementaci aplikace. Jedná se o open–source systém řízení databáze, ze kterého můžeme získávat data pomocí SQL dotazů. Využívá relační databázový model, tzn., že vytváří vztahy mezi jednotlivými tabulkami. Tabulka má pevně dané atributy (sloupce), kterým je zároveň určena jejich velikost a datový typ [7].

MySQL patří do rodiny SQL (Structured Query Language), která se jakožto starší oproti NoSQL (Not only SQL) soustřeďuje na redukci duplicity dat k dosažení menších paměťových nároků. NoSQL se oproti SQL databázím skládají z tzv. dokumentů ve formátu JSON, které jsou snadněji škálovatelné. MySQL je šířeno pod bezplatnou licencí GPL [7].

Pro PHP bylo vytvořeno i mnoho frameworků, tedy platforem, poskytujících knihovny funkcí, které jsou v PHP často využívány. Tím dochází k minimalizaci časové náročnosti spojené se psaním kódu. Jsou jednodušší pro údržbu a také poskytují dobrý základ v oblasti zabezpečení aplikace. Mezi nejpoužívanější frameworky patří Laravel (viz kapitola 1.1.2), Symfony nebo CodeIgniter. PHP frameworky také typicky podporují architekturu MVC (viz kapitola 1.2.1) [8].

1.1.2 Framework Laravel

Laravel je bezplatný open-source framework poskytující sadu nástrojů pro vytváření moderních webových aplikací v jazyce PHP. Cílem jeho vývojářů bylo vytvořit jednoduchou a elegantní syntaxi pro rychlý a efektivní vývoj webových aplikací. Framework Laravel je šířen pod MIT licencí. Poskytuje rozsáhlou dokumentaci spolu s edukačními návody v podobě videí, což činí z Laravelu velice dobře pochopitelný a naučitelný framework [9].

Laravel usnadňuje běžné úlohy používané u většiny webových aplikací, jako je autentizace a autorizace uživatele, směrování, práce s databází a ukládání do mezipaměti (cache). Při zakládání nového projektu automatizovaně vytvoří definovanou adresářovou strukturu, a zároveň plně funkční vzorovou aplikaci. Při využití některého ze dvou startovacích sad – Laravel Breeze a Laravel Jetstream, je automaticky vygenerována funkční webová aplikace s možností registrace a přihlášení uživatele. V praktické implementaci této práce byla využita sada Laravel Breeze. Sada Laravel Jetstream poskytuje i propracovanější front-end [9].

Kromě zmíněných funkcí poskytuje Laravel i výkonné databázové nástroje včetně ORM (Object Relational Mapper) s názvem Eloquent, usnadňující komunikaci s databází tím, že vytváří objektově orientovanou vrstvu mezi relační databází a objektově orientovaným programovacím jazykem (v tomto případě PHP) bez nutnosti vytvářet dotazy SQL [10]. Při používání nástroje Eloquent má každá databázová tabulka vytvořený odpovídající Model (viz kapitola 1.2.1), který je používán k interakci s danou tabulkou. Kromě načítání záznamů z databázové tabulky umožňují modely Eloquent také vkládat, aktualizovat a mazat záznamy z tabulky [9].

Obsahuje také nástroj příkazové řádky Artisan, pomocí něhož mohou vývojáři zavádět nové modely, kontroléry, pracovat s cache pamětí, vytvářet databázové tabulky a další komponenty aplikace, což urychluje celkový vývoj aplikace. Artisan existuje v kořenovém adresáři aplikace jako skript `artisan` [9].

Architektura samotné aplikace využívá architekturu MVC (viz kapitola 1.2.1), podle kterého je definována i adresářová struktura, která mimo jiné obsahuje adresáře Models, Views a Controllers (více viz kapitola 5.2). Stylizace výchozí aplikace je zajištěna CSS frameworkem Tailwind [9]. Při praktické implementaci webové aplikace ale tento framework nebyl použit a byl nahrazen vlastními CSS styly.

1.2 Architektura webové aplikace

Architektura webové aplikace je mechanismus určující způsob komunikace jednotlivých komponent aplikace. Uživatelé sítě Internet jsou zahlceni různými zdroji těch

stejných informací. V poslední době se proto vývojáři orientují hlavně směrem uživatelské přívětivosti. Klíčem k dosažení co nejlepších výsledků v této oblasti je předem připravená architektura webové aplikace, její stabilita a škálovatelnost.

Dalším důvodem, proč se vůbec přemýšlí nad architekturou a ne jen nad naprogramováním aplikace je nejspíše ten, že jednotlivé technologie mají ranou fázi vývoje za sebou a vedle „základní výbavy“ se soustřeďují i na optimalizace, které výrazně souvisí s vhodnou volbou architektury aplikace tak, aby se předcházelo problémům již ve stádiu vývoje [11].

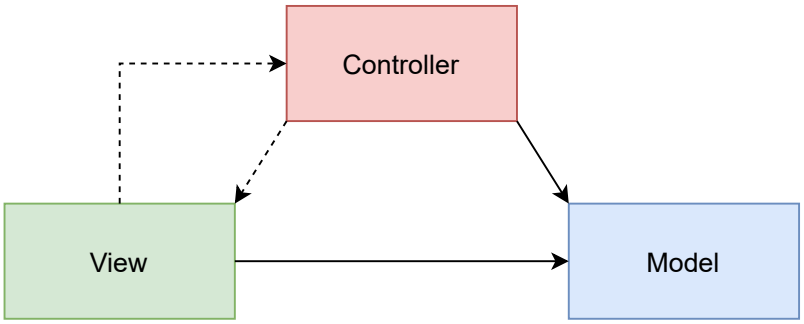
Dnes je k dispozici mnoho technik, které pomohou se smysluplnou a funkční strukturou webové aplikace. Kapitoly 1.2.1 a 1.2.2 se zabývají architekturami, které usnadňují orientaci v aplikaci.

1.2.1 Model-View-Controller

Architektura Model-View-Controller (dále jen MVC) definuje způsob přehledné organizace kódu. Hlavní myšlenkou MVC je rozdělení aplikace do tří částí, kde každá má svůj účel a vztah s těmi ostatními. Mezi tyto části patří Model, View a Controller. Model reprezentuje data a business logiku aplikace, View zobrazuje uživatelské rozhraní a Controller má na starost tok událostí v aplikaci a obecně aplikační logiku [11].

Díky zmíněnému rozdělení na části je možné aplikaci vyvíjet rychleji, protože na ní může pracovat hned několik vývojářů zároveň - jeden vyvíjí View, druhý zase Model atd. Je možné vyvíjet více View pro jeden Model, např. poskytovat další možnosti přístupu k datům aplikace prostřednictvím různých aplikačních rozhraní. Také je možné díky podpoře JavaScriptových knihoven používat asynchronní techniky, což je velká výhoda například při rychlosti načítání aplikace. V neposlední řadě je kód aplikace pro vývojáře lépe čitelný a srozumitelný [11].

MVC si můžeme detailněji představit na příkladu aplikace pro vytváření seznamu úkolů. Model v této aplikaci definuje, co je úkol, a že seznam je kolekce těchto úkolů. View na druhou stranu říká, jak úkoly a seznamy vypadají vizuálně, např. jakým písmem či barvou bude napsán text. Controller se potom stará o to, jak uživatel přidá úkol do seznamu nebo jak označí jiný ukol za splněný - Controller propojí tlačítko pro přidání ve View s Modelem tak, že po kliknutí Model přidá nový úkol do databáze [12].



Obr. 1.2: Návaznost jednotlivých komponent architektury MVC

Vztah jednotlivých částí architektury MVC je vidět na obrázku 1.2. Jak naznačují šipky, v architektuře MVC existují pouze dvě přímé vazby. Controller má přímý odkaz na Model, aby mohl upravit jeho data. View má přímý odkaz na Model, aby mohl jeho data zobrazit. Přímá vazba mezi Modelem a ostatními částmi nikdy nesmí existovat - to by bylo zásadní porušení konceptu architektury MVC [11]. Webová aplikace vyvíjená v rámci této práce bude stavět právě na této architektuře.

1.2.2 REST

Representational State Transfer (REST) je termín, který v roce 2000 vymyslel Roy Fielding. Jedná se o styl architektury pro navrhování webových aplikací. REST nevynucuje žádné postupy týkající se toho, jak by měla být aplikace implementována, pouze popisuje pravidla, která definují tzv. RESTful webovou službu:

- **jednotné rozhraní** – jakmile se vývojář seznámí s jedním RESTful systémem, měl by být schopen postupovat podobně i u ostatních,
- **klient–server** – klientské aplikace a serverové aplikace se musí vyvíjet samostatně bez vzájemné závislosti,
- **bezestavovost** – každý požadavek obsahuje všechny informace potřebné k jeho úspěšnému zpracování,
- **zálohovatelnost** – správně řízené ukládání do mezipaměti (cache) částečně nebo zcela eliminuje některé interakce mezi klientem a serverem, což dále zlepšuje škálovatelnost a výkon webové aplikace,
- **vrstevnatost** – skládání a propojování služeb poskytujících službu v rámci zvýšení variability systému,
- **kód na vyžádání** – v případě potřeby může server poslat spustitelný kód pro podporu části své aplikace.

REST si lze představit jako přenos reprezentace zdrojů, přičemž zdrojem může být téměř cokoli, např. obrázek, video, kniha, apod. Každý zdroj má specifický identifikátor URL (Uniform Resource Locator), na jehož základě klient posílá své požadavky na server a server v odpovědi posílá reprezentaci požadovaného zdroje. Slovem reprezentace rozumíme formát čitelný pro člověka - HTML, obrázek, JSON (JavaScript Object Notation) nebo XML (Extensible Markup Language). REST definuje čtyři základní metody, známé pod zkratkou CRUD (Create Retrieve Update Delete) pro přístup ke zdrojům, odpovídajícím metodám protokolu HTTP - POST, GET, PUT, DELETE [13].

1.3 Návrhové vzory webových aplikací

Vždy je důležité zvolit nejvhodnější architekturu s ohledem na různé faktory, jako je logika aplikace, vlastnosti, funkce, obchodní požadavky atd. Správná architektura také definuje zaměření a účel webové aplikace jako celku. Existuje několik typů, které jsou probrány v následujících kapitolách.

1.3.1 Jednostránkové webové aplikace

Zkráceně SPA (Single-Page Application) označuje webovou stránku nebo aplikaci, která načte veškerý obsah i funkcionalitu najednou - při používání již nepotřebuje dodatečné údaje ze serveru. Webový prohlížeč obdrží od serveru jeden HTML dokument a další data načítá pomocí frameworků JavaScriptu, jako je Angular, React a další. Mezi takto navržené webové stránky se řadí například Gmail, Facebook nebo Twitter [14].

SPA aplikace mají jednu významnou výhodu - jsou velmi uživatelsky přívětivé, protože zde odpadá opakované načítání webových stránek. Na druhou stranu, jejich nevýhodou jsou problémy s SEO (Search Engine Optimalization) optimalizacemi, tzn. optimalizacemi pro vyhledávání v internetových vyhledávačích, pro které je obtížnější takto navržené webové aplikace indexovat [14].

1.3.2 Vícestránkové webové aplikace

Vícestránkové aplikace označované také jako MPA (Multi-Page Application) jsou na Internetu populárnější, protože v minulosti byly všechny webové stránky založeny na architektuře MPA [14].

V dnešní době se společnosti rozhodují pro MPA, pokud jsou jejich webové stránky obsahově rozsáhlejší (jako například eBay). Taková řešení znovu načítají

webovou stránku, aby se načetly nebo odeslaly informace z nebo na server prostřednictvím webových prohlížečů uživatelů. Takto navržené aplikace jsou mnohem lépe optimalizovatelné v rámci SEO. Do nevýhod patří složitější implementace responzivního designu [14].

1.3.3 Architektura mikroslužeb

V klasických webových aplikacích se využívá přístup tzv. monolitické architektury, kdy spolu komunikují front-end, back-end a databáze k zajištění služby. Pokud vývojář potřebuje jednu z těchto částí upravit, zpravidla musí učinit změny i v dalších částech aplikace [14].

Pokud jde o mikroslužby, tento přístup umožňuje vývojářům vytvořit webovou aplikaci ze sady menších služeb. Vývojáři vytvoří a nasadí každou komponentu zvlášť. Tato architektura se tak stává vhodnou pro velké a složité projekty, protože každou službu lze měnit nezávisle na ostatních. Například, pokud by bylo potřeba aktualizovat platební logiku, nemusela by se zastavit kompletně celá aplikace [14].

1.3.4 Webová aplikace v cloudu

Tento typ architektury webových aplikací umožňuje vývojářům využívat cloudovou infrastrukturu od poskytovatelů služeb třetích stran, jako jsou Amazon nebo Microsoft [14].

Aby mohla webová aplikace fungovat na Internetu, měli by vývojáři spravovat serverovou infrastrukturu (ať už virtuální nebo fyzickou), operační systém a další procesy související s hostováním serveru. Poskytovatelé cloudových služeb nabízejí virtuální servery, které dynamicky řídí přidělování výpočetních prostředků. Jsou proto mnohem lépe škálovatelné a tak lépe připravené například na výrazný nárůst provozu, na který nejsou fyzické firemní servery často stavěny [14].

1.3.5 Progresivní webové aplikace

Jedním z hlavních trendů ve vývoji webových aplikací posledních let jsou progresivní webové aplikace. Jedná se o webová řešení navržená tak, aby se na mobilních zařízeních chovala jako nativní aplikace. PWA (Progressive Web Application) nabízejí push notifikace, přístup bez připojení k internetu a možnost instalace aplikace na domovskou obrazovku [14].

K vytvoření takové aplikace jsou používány webové technologie jako HTML, CSS a JavaScript. Pokud aplikace potřebuje přístup k funkcím zařízení, jsou využity další rozhraní API - NFC API, Geolocation API, Bluetooth API a další [14].

2 PrestaShop

PrestaShop je bezplatná open-source platforma implementovaná v jazyce PHP pro vytváření webových aplikací v podobě internetových obchodů s řadou užitečných funkcí, která může být použita k provozování jak v cloudových službách, tak na webovém hostingu. Jeho historie sahá až do roku 2005, kdy PrestaShop vznikl jako studentský projekt na vysoké škole EPITECH v Paříži. V současné době tuto platformu používá přes 300 000 e-shopů po celém světě (z toho zhruba 7 000 v České republice) a je k dispozici v 65 různých jazykových mutacích [15]. PrestaShop, spolu s jeho největšími konkurenty jako Shopify, Magento nebo OpenCart, vydělávají na zprostředkování jednoduchého a rychlého způsobu založení e-shopu [16].

Je samozřejmostí, že takové řešení bez potřeby vývoje vlastního software na míru je mnohem méně finančně nákladné. Pomocí přídavných modulů navíc umožňují rozšířit funkcionalitu základního systému, jako například propojení s platební bránou, emailové služby, designové editory apod. Tato skutečnost je důvodem, proč jsou tyto platformy tolik využívané – díky jejich jednoduché instalaci a modularitě jsou provozovatelé schopni do několika hodin spustit funkční e-shop. Na druhou stranu, pokud se v systému nebo modulu vyskytne bezpečnostní chyba, je velmi pravděpodobné, že jich bude v malé chvíli velké množství zkompromitováno [16].

2.1 Architektura

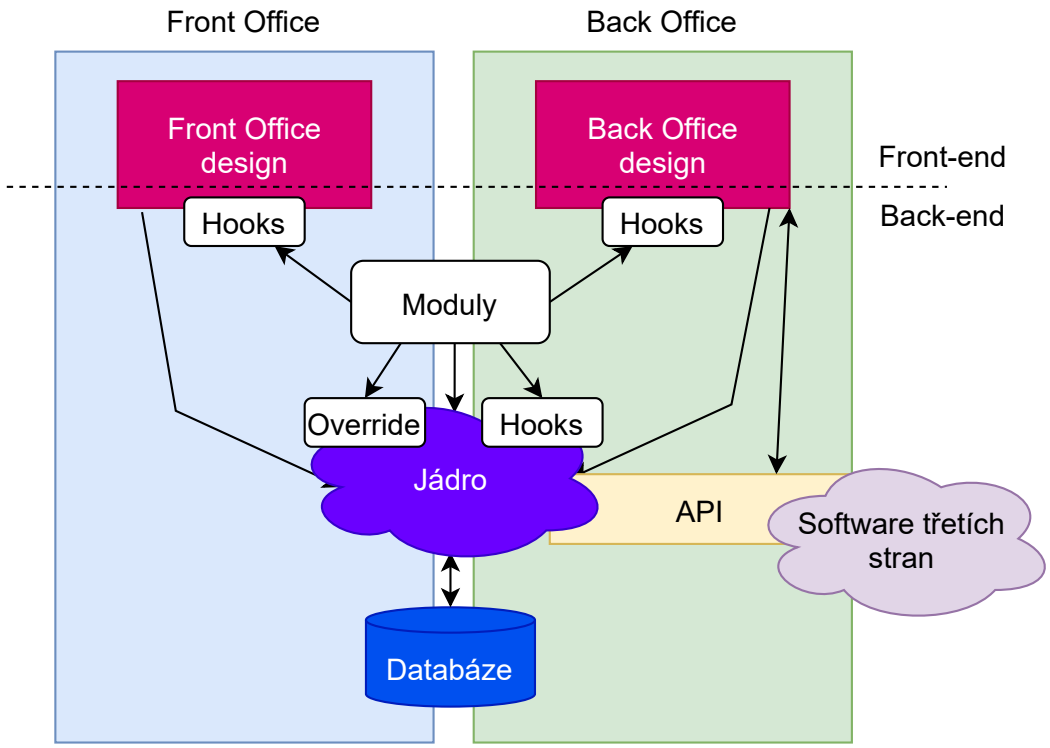
PrestaShop byl vytvořen jako monolitická webová aplikace podle tradičních objektově orientovaných postupů programovacího jazyka PHP (od verze 5.6 a vyšší). Původně byl založen na vlastním frameworku, ale od verze 1.7 je postupně převáděn na framework Symfony, využívající architekturu MVC, viz kapitola 1.2.1 [16].

Architekturu PrestaShopu lze rozdělit na dvě hlavní logické části, které jsou na obrázku 2.1 znázorněny jako modrý a zelený sloupec:

- **Front Office** – stránka obchodu zobrazovaná zákazníkům,
- **Back Office** – místo, kde obchodníci spravují svůj obchod.

Každou z těchto sekcí lze samu o sobě rozdělit na dvě části, které jsou společné všem webovým aplikacím. Toto rozdělení je znázorněno pomocí přerušované vodorovné čáry:

- **Front-end** – část, která běží v prohlížeči,
- **Back-end** – běží na serveru.



Obr. 2.1: Architektura platformy PrestaShop

Back Office se dále skládá z dalších komponent – databáze, modulů a jádra. Stejně jako většina tradičních webových aplikací je i PrestaShop do značné míry založen na databázi. To znamená, že jsou v ní uložena všechna data. Bez ohledu na to, zda se používají v Back Office nebo Front Office, je databáze jediným zdrojem dat [16].

Jádro obsahuje obchodní logiku aplikace. Zahrnuje modely, kontroléry, pomocné třídy a podobně. Zde můžeme vidět provázanost s MVC architekturou (viz kapitola 5.2) [16].

Moduly jsou nezávislé volitelné balíčky, které rozšiřují a přizpůsobují PrestaShop mnoha způsoby. S jádrem komunikují buď tak, že se připojí k bodům rozšíření pomocí tzv. hooks, které jsou umístěny v celém kódu aplikace nebo tak, že přímo nahradí komponenty jádra svými vlastními [16].

Design webové aplikace je v rámci PrestaShopu realizován pomocí šablonovacího systému Twig, který poskytuje speciální syntaxi pro oddělení PHP a HTML částí a je distribuován jako součást frameworku Symfony (více v kapitole 2.1.1). V PrestaShopu lze ovlivnit jak vzhled Back Office tak Front Office [16].

2.1.1 Rozdíl mezi PrestaShop verzemi 1.6 a 1.7

PrestaShop je aktuálně ve verzi 1.7, přesto jsou po celém světě využívány verze 1.6, 1.5, dokonce i 1.4 a níž, které jsou samozřejmě neaktuální, pomalé a nezabezpečené proti aktuálním hrozbám. Verze 1.4 byla přitom vydána v roce 2011, verze 1.5 v roce 2012. Tato kapitola je zaměřena na porovnání posledních dvou z nich [17].

Verze 1.6 byla vydána v roce 2014. Dnes působí ve své aktuální verzi 1.6.1.24 jako odladěný a spolehlivý systém. Z technického hlediska ale tato verze nedržela krok s vyvíjejícím se programovacím jazykem PHP, na kterém je PrestaShop postaven. Tato verze podporuje PHP nejvýše ve verzi 7.1. Proto bylo zejména kvůli výkonnostním a bezpečnostním požadavkům nutné začít vyvíjet novou verzi podporující aktuální verze jazyka PHP. Samozřejmě se zveřejněním nové verze PrestaShopu 1.7 klesá i podpora ze strany vývojářů modulů potřebných pro plnohodnotný chod systému. Z dlouhodobého hlediska je tedy výhodnější přejít na nejnovější verzi [17].

V reakci na shrnuté nedostatky vyšla v roce 2016 verze 1.7. Jeho hlavním vylepšením, od kterého se ostatně odvíjí všechny další, je využití již zmíněného PHP frameworku Symfony. Jedná se o webový aplikační framework pro vývoj webových aplikací, vycházející z architektury MVC, obsahující znovupoužitelné PHP komponenty pro jednodušší a rychlejší vývoj. Celý framework je z velké části inspirován jinými webovými aplikačními frameworky jako Ruby on Rails, Django a Spring [18][19].

Toto řešení zaručuje ty nejaktuálnější funkce a verze jazyka PHP, protože se opět jedná o open-source projekt mohutně využívaný po celém světě. Jedná se o přímého konkurenta frameworku Laravel (viz kapitola 1.1.2)[18].

Nejnovější verze systému PrestaShop je tak mnohem rychlejší než předchozí a samozřejmě poskytuje vyšší míru ochrany vůči kybernetickým hrozbám, na které byly verze předchozí zranitelné. Lepší je samozřejmě i z uživatelského, potažmo marketingového hlediska. PrestaShop 1.7 výrazně zjednodušuje práci vývojářům a designerům díky výchozí šabloně, která je dostupná zdarma a neobsahuje žádné předem definované styly. Obsahuje také vylepšené funkce, jako například novou stránku pro vkládání produktů, opravené chyby při nákupním procesu zákazníka a kompletně přestavený Back Office (viz kapitola 2.1)[19].

2.2 Bezpečnost

Nastavení a spuštění PrestaShopu je sice snadné, ale je třeba dát pozor na jeho zabezpečení. Internetové obchody obvykle spravují citlivé informace, jako jsou osobní údaje uživatelů nebo údaje o kreditních kartách. Pro firmy je obrovskou odpovědností s takovými citlivými údaji bezpečně nakládat. Statistiky hackerských útoků

na PrestaShop podle zdroje [20] ukazují, že až 40% e-shopů je zranitelných vůči Cross-Site Scripting útokům. 19% je zranitelných vůči Remote Code Execution a 6,7% vůči SQL Injection. Kromě zmíněných útoků také běží velké množství webů se systémem PrestaShop na zastaralých verzích.

Cross-Site Scripting (XSS) je typ zranitelnosti webové aplikace, při níž jsou škodlivé skripty vkládány do jinak neškodných a důvěryhodných webových stránek. Útočník takovýmto způsobem donutí webovou aplikaci poskytovat skript dalším koncovým uživatelům. Webové prohlížeče nijak neověřují soubory, které obdrželi, protože pocházejí z důvěryhodného zdroje a útočníkův skript spustí. Útočník tak získává přístup ke cookies, relacím a dalším citlivým informacím legitimních uživatelů, případně může měnit obsah webové stránky [21].

SQL Injection je jednou z nejčastějších hrozeb pro databázový systém využívaný webovou aplikací, při které útočník zadává škodlivé SQL příkazy ve vstupním poli formuláře aplikace, aby získal přístup ke zdrojům nebo provedl změny dat uložených v databázi. Důsledkem může být zničena funkčnost a důvěrnost webové aplikace. Tento útok je umožněn nedostatečnou validací zmíněných vstupů [22].

Remote Code Execution (RCE) je koncept útoku, který popisuje formu kybernetického útoku, při němž může útočník ovládat činnost cizího výpočetního zařízení. K RCE dochází v případě, kdy je do hostitelského počítače stažen škodlivý malware [23]. U PrestaShopu tuto zranitelnost zpřístupnil modul Responsive Mega Menu Pro [24] , který umožnil předávat příkazy v v rámci HTTP GET požadavku.

2.2.1 Současná situace

I v dnešní době je bezpečnost systému PrestaShop poměrně špatná. Důvodem je zejména již zmíněný problém aplikace odolávat proti útokům SQL Injection. SQL dotazy totiž nejsou posílány na databázi jako tzv. prepared statements, ale skládají se pomocí konkatenace, což zvyšuje pravděpodobnost, že vývojáři zapomenou hodnoty posílané na databázi ošetřit, a vznikne tak prostor pro provedení SQL injekce. Toto se týká starších částí systému PrestaShop a modulů třetích stran, které ještě nebyly převedeny do frameworku Symfony.

Prepared statements se skládají ze dvou fází: přípravy a provedení SQL dotazu. Ve fázi přípravy je databázovému serveru odeslána šablona příkazu. Server provede kontrolu syntaxe a inicializuje své vnitřní prostředky pro pozdější použití. Po přípravě následuje provedení, v rámci něhož klient odesílá hodnoty dotazovaných parametrů. Server provede příkaz s přijatými parametry pomocí dříve vytvořených vnitřních zdrojů a odešle odpověď. Potenciálnímu útočníkovi je tak znemožněno vkládat do SQL požadavku škodlivé řetězce, protože takovéto požadavky nebudou databází provedeny [25].

S migrací na modernější technologie se bezpečnost systému PrestaShop výrazně zlepšuje (viz kapitola 2.1.1). Přesto se nevyskytuje mnoho nástrojů pro ověření jeho bezpečnosti. Existuje pouze několik modulů PrestaShopu poskytujících určitou míru zabezpečení tím, že se zaměřují na určitou zranitelnost, např. útoky hrubou silou, ošetřování vstupů, útoky zamezení služby nebo detekci malwaru.

Byly nalezeny dva nástroje zajišťující rozsáhlejší skenování bezpečnosti webových aplikací postavených na systému PrestaShop - Astra [26] a HTTPCS [27]. Zabývají se vytvářením firewallových pravidel, skenováním malwaru, bezpečnostními audity a hardeningem. Přesto ale není ani jeden z nástrojů zaměřen přímo na platformu PrestaShop, ale obecně na více CMS systémů, což může značit menší přesnost prováděných testů, jelikož nejsou optimalizovány pro konkrétní platformu.

Služba zajišťující penetrační testování nalezena nebyla. Tato skutečnost otevírá dveře útočníkům, kteří mají díky absenci takovýchto služeb větší šanci na úspěšné kompromitování webové aplikace.

3 Penetrační testování

Penetrační testování lze definovat jako legální a autorizovaný pokus o nalezení a úspěšné zneužití zranitelností počítačových systémů za účelem posouzení jejich bezpečnosti. Závěr penetrační testování vždy končí konkrétními doporučeními pro řešení a opravu bezpečnostních chyb, které byly během testu odhaleny [28].

Motivací pro firmy provádět penetrační testy je vyhodnocení celkového zabezpečení jejich IT infrastruktury a odolnosti vůči reálným kybernetickým útoků, které mohou mít za následek obrovské finanční, datové i materiální ztráty. Společnost může mít v jedné oblasti spolehlivé bezpečnostní protokoly, ale v jiné oblasti se mohou objevit nedostatky [28].

Hlavním rozdílem provádění penetračních testů od reálných kybernetických útoků je ten, že je před jeho začátkem podepsána smlouva mezi testerem a majitelem systému, specifikující požadavky a připomínky obou stran. Zejména je popsán rozsah penetračních testů. Je také důležité zmínit, že penetrační testy provádí penetrační tester, často označován jako etický hacker [28].

3.1 Dělení penetračních testů

Penetrační testy lze dělit hned podle několika kritérií. Nejvíce používané je dělení do čtyř skupin. První skupinou je obecné dělení na interní a externí testy: [28]

- **Interní testování** – testy jsou prováděny zevnitř testovaného systému, aby napodobily možnosti uživatele, který nějakým způsobem získal přístup do vnitřní sítě společnosti. Často se zde také setkáváme se scénářem nespokojeného zaměstnance. Tento typ testování se soustředí na odhalování chyb, zranitelností, špatné nastavení autentizace. V praxi je majitelem systému umožněn vstup testera do vnitřní sítě.
- **Externí testování** – testy jsou realizovány zvenčí a představují vnější hrozbu, tedy útoky hackerů z internetu. Útočí se na servery, webové aplikace a další služby dostupné komukoliv přistupujícímu z internetu. V této práci se budeme zabývat právě tímto druhem penetračního testování.

Druhá skupina dělení se týká znalosti testovaného systému, kde rozlišujeme Black-box, White-box a Grey-box testování: [28]

- **Black-box** – na testovaný systém je nahlíženo jako na tzv. černou skříňku, u níž jsou známy pouze její vstupy a potenciální výstupy. Vnitřní struktura systému není známa. Tato metoda je typická pro hackery, kteří mají k dispozici

pouze veřejně dostupné informace (např. název domény serveru), které jsou podrobeny dalšímu zkoumání.
- **White-box** – k dispozici jsou všechny dostupné znalosti o testovaném systému, tzn. informace o topologii systému, aktuální konfigurace, různé přístupy k datům, nastavení síťových prvků, zdrojové kódy, dokumentace atd. Takto vedené testování umožňuje odhalení chyb v kratším čase a díky větší znalosti systému také přináší vyšší pravděpodobnost nalezení bezpečnostní chyby.
- **Grey-box** – kombinace obou výše zmíněných typů. Tester má pouze základní znalosti o systému (znalost vnitřní struktury sítě), které se snaží co nejvíce zneužít. Výhodou je, že se tester může zaměřit přímo na rizikovější části systému a neztrácet čas skenováním celého systému.

Další způsob dělení je podle způsobu provedení penetračního testu na manuální, automatizované a semiautomatizované testy: [28]

- **Manuální testy** – tester provádí testy manuálně. Výhodou takového postupu je optimalizace testů přímo pro konkrétní podmínky. Nevýhodami jsou naopak nutná rozsáhlá znalost testované oblasti, vytváření vlastních testovacího postupů a časová náročnost.
- **Automatizované testy** – tester využívá nástroje pro automatizované testování vyvíjené profesionály v oboru. Tester se musí pouze naučit s nástroji pracovat. Výhodou je rychlost provedení penetračního testování, nevýhodou může být nemožnost testovat určité typy zranitelností.
- **Semiautomatizované testy** – jedná se o kombinaci automatických a manuálních testů využívající výhod obou metod.

Čtvrtým možným rozdělením testů je rozdělení podle cíle, které dnes již definuje samostatné disciplíny penetračního testování. Jedná se o testováni síťové infrastruktury, webových aplikací a mobilních aplikací [28].

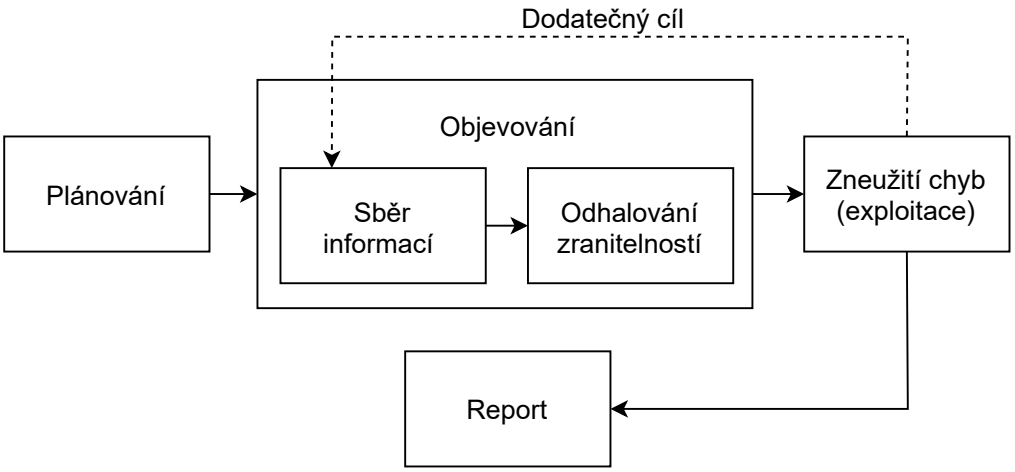
3.2 Metodologie testování

Pro samotné penetrační testování nestačí pouze znalost bezpečnostních nástrojů, sítí apod. Je také důležité vědět, jak tyto nástroje a znalosti využít v rámci celkového schématu penetračního testování. Proto byly vytvořeny metodiky, které sumarizují kroky potřebné pro provedení kompletního bezpečnostního testování, například: [28]

- Open Source Security Testing Methodology Manual (OSSTMM)
- Information Systems Security Assessment Framework (ISSAF)

- NIST 800-115 – Technical Guide to Information Security Testing and Assessment
- Open Web Application Security Project (OWASP)
- OSSTMM Web Application Methodology

Metodiky nejsou vždy zcela jednotné, ale základní postup je stejný. Počet jednotlivých kroků se pohybuje od čtyř do sedmi. Na obrázku 3.1 je znázorněna metodika, která zahrnuje celkem pět kroků [28].



Obr. 3.1: Metodologie testování

První fáze se nazývá plánování. To si klade za cíl stanovit všechny organizační požadavky, vytvoření časového plánu a stanovení podrobných cílů, na které budou penetrační testy zaměřeny. To vše samozřejmě záleží na schopnostech testera a finančních prostředcích zadavatele [28].

Druhou fází je sběr informací, kdy se tester snaží získat co nejvíce dat o cílovém systému, potažmo síti. Pro příklad je možné uvést rozsah IP adres, otevřené porty, síťové služby apod. Jsou zde využívány nástroje jako nmap, zenmap, nebo i techniky jako tzv. sociální inženýrství, což je způsob manipulace s lidmi za účelem získání citlivých informací [28].

Třetí fáze se zabývá odhalováním zranitelností. Při hledání zranitelností se porovnávají nalezené informace o systému a síti s databází známých zranitelností jako například CVE [29]. Také se kontrolují chybné konfigurace zařízení a služeb. Výsledkem je seznam obsahující zranitelnosti a jejich stupeň závažnosti. Pro tento účel se používají nástroje jako Nessus nebo Exploit DB [28].

Čtvrtou fází je exploitace, neboli zneužití nalezených zranitelností pro prolomení bezpečnostních mechanismů. V této fázi je možné, že budou objeveny další dříve ne-

přístupné služby. V takovém případě je nutné znovu provést fázi čtyři. Nejznámějším nástrojem pro exploitaci je Metasploit Framework [28].

Poslední fází testování je vypracování závěrečné zprávy neboli reportu, který obsahuje shrnutí a předběžnou analýzu provedených penetračních testů. Cílem je předložit zadavateli reálné řešení, které povede ke zvýšení bezpečnosti systému [28].

3.2.1 OWASP

OWASP (The Open Web Application Security Project) je celosvětová nezisková organizace, která se zabývá zlepšováním bezpečnosti webových aplikací. Jejím cílem je, aby všichni měli neomezený přístup k informacím o bezpečnostních hrozbách, na základě kterých by mohli provádět bezpečnostní opatření. OWASP provozuje přes 100 aktivních projektů [30].

Nejznámějším z projektů je OWASP Top Ten. Tento dokument je standardem pro informovanost vývojářů a zabezpečení webových aplikací dostupný zdarma. Představuje seznam nejkritičtějších bezpečnostních rizik pro webové aplikace, na kterém se podílejí bezpečnostní experti z celého světa. Pro rok 2021 je jmenováno těchto deset nejrozšířenějších zranitelností definovaných ve zdroji: [31]

- A01 – Disfunkční kontrola přístupu,
- A02 – Chyby v implementaci kryptografických systémů,
- A03 – Injekce,
- A04 – Nebezpečný design aplikace,
- A05 – Špatná konfigurace bezpečnostních opatření,
- A06 – Zranitelné a zastaralé komponenty,
- A07 – Selhání identifikace a autentizace,
- A08 – Selhání integrity softwaru a dat,
- A09 – Nedostatečné logování a monitorování,
- A10 – Server-Side Request Forgery.

4 Vývojové prostředí

Tato kapitola se zabývá topologií vývojového prostředí, využitými komponenty a jejich nastavením. Dále představuje testovací prostředí, které je možné využít pro co nejjednodušší otestování vlastní aplikace.

Vývojové prostředí je realizováno v operačním systému MacOS Monterey 12.0.1. Pro naplnění zadání je potřeba spustit dvě webové aplikace. Jednu pro implementovanou webovou aplikaci, která nese název PrestaCure, a druhou pro testovací webovou aplikaci využívající systém PrestaShop. Toho lze dosáhnout buďto použitím dvou serverů, nebo využitím konceptu virtuálních hostitelů, tedy hostování více webových stránek (aplikací) na jednom webovém serveru.

4.1 MAMP

V této práci bude využit zmíněný koncept virtuálních hostitelů na serveru Apache ve verzi 2.4.46, který je provozován pomocí software MAMP. Jedná se o bezplatné prostředí pro provoz lokálního serveru, které lze nainstalovat na systémy MacOS a Windows. MAMP také poskytuje databázový server MySQL verze 5.7.34, volbu mezi webovými servery Apache nebo Nginx a podporu programovacích jazyků PHP, Python, Perl a Ruby [32].

Nastavení virtuálních hostitelů se na webovém serveru Apache provádí v konfiguračním souboru `../MAMP/conf/apache/httpd.conf`. V něm je potřeba definovat následující záznam:

Výpis 4.1: Konfigurace souboru `httpd.conf` na webovém serveru Apache

```
1  NameVirtualHost *
2
3  <VirtualHost *>
4      DocumentRoot "/path/to/webapp/one"
5      ServerName prestacure.cz
6  </VirtualHost>
7
8  <VirtualHost *>
9      DocumentRoot "/path/to/webapp/two"
10     ServerName prestashop.cz
11 </VirtualHost>
```

Pro lokální překlad specifikovaných doménových jmen na `localhost` je potřeba editovat soubor `/etc/hosts`:

Výpis 4.2: Konfigurace souboru `/etc/hosts`

```
1 127.0.0.1       prestashop.cz
2 127.0.0.1       prestacure.cz
```

4.2 Webová aplikace PrestaCure

Webová aplikace PrestaCure je praktickým výstupem této práce, která svým uživatelům umožní otestovat jejich webové aplikace postavené na platformě PrestaShop (viz kapitola 2). Detailně je popsána v kapitole 5.

4.3 PrestaShop

Pro vývoj základních funkcionalit aplikace PrestaCure sloužila webová aplikace využívající systém PrestaShop ve verzi 1.7.8.1 stažená z oficiálních stránek PrestaShopu [16]. Ve výchozím stavu obsahují všechny verze tohoto systému funkční e-shop se základním nastavením a několika produkty. Instalace je velmi jednoduše zpracována a hotova během několika minut.

4.4 Testovací prostředí

Pro co nejjednodušší otestování aplikace PrestaCure a případně i další vývoj byl sestaven virtuální stroj `PrestaCure - Ubuntu 64-bit 20.04.4`, dostupný na cloudovém úložišti (odkaz naleznete v příloze D). Byl vytvořen v multiplatformním virtualizačním open-source nástroji VMware. Kromě vlastní aplikace PrestaCure obsahuje i tři webové aplikace se systémem PrestaShop - každý v různé verzi (1.4.7, 1.5.0.17, 1.6.1.16 a 1.7.8.5) a nástroj Composer, určený pro správu knihoven a dalších zdrojů v PHP pro lepší práci s frameworkem Laravel [33]. Přihlašovací údaje jsou ke všem webovým aplikacím na virtuálním stroji stejné:

Výpis 4.3: Výchozí přihlašovací údaje

```
1 // Uživatel Admin v aplikaci PrestaCure
2 admin@email.cz:password
3 // Do všech aplikací se systémem PrestaShop
4 admin@email.cz:password
```

Pro správnou funkčnost všech webových aplikací bylo nutné provést několik úprav. Zejména proto, že zmíněné různé verze PrestaShopu podporují různé verze programovacího jazyka PHP a nelze využít jednu verzi pro všechny. Proto bylo nutné vytvořit webový server, který bude podporovat tuto funkcionalitu.

Na virtuálním stroji byl nainstalován webový server Apache ve verzi 2.4.41, který používá koncept virtuálních hostitelů ke správě více domén v jedné instanci. K němu byl doinstalován software PHP-FPM používající démona ke správě více verzí jazyka PHP v jedné instanci. Tento software vytváří vlastní konfigurační soubory v adresáři `/etc/apache2/sites-available/` pro virtuální hosty se specifikací verze PHP, které jsou načítány webovým serverem [34]. Vypadají např. takto:

Výpis 4.4: Konfigurační soubor pro virtuální hosty v PHP-FPM

```
<VirtualHost *:80>
    ServerName prestashop-1-7.loc
    DocumentRoot /var/www/prestashop-1-7
    DirectoryIndex index.php

    <Directory /var/www/prestashop-1-7>
        Options Indexes FollowSymLinks MultiViews
        AllowOverride All
        Order allow,deny
        allow from all
    </Directory>

    <FilesMatch \.php$>
        SetHandler "proxy:unix:/run/php/php7.4-fpm.sock|fcgi
           ://localhost"
    </FilesMatch>

    ErrorLog ${APACHE_LOG_DIR}/prestashop-1-7_error.log
     CustomLog ${APACHE_LOG_DIR}/prestashop-1-7_access.log
        combined
</VirtualHost>
```

Společně tak vytváří možnost hostování více webových aplikací, z nichž každá používá jinou verzi PHP, na jednom serveru. Tato kombinace je také efektivnější a méně nákladná na zdroje, než hostování každé aplikace v její vlastní instanci.

Jak aplikace se systémem PrestaShop, tak aplikace PrestaCure potřebují ke své funkčnosti databázi. Podporovanou variantou je databáze MySQL (viz kapitola 1.1.1).

Kvůli zpětné kompatibilitě ovšem nebylo možné použít její nejnovější verzi MySQL 8.0, ale starší verzi MySQL 5.6.

Aby byly aplikace dostupné v uživatelsky přívětivějším formátu, byly v souboru `/etc/hosts` vytvořeny tyto záznamy:

Výpis 4.5: Konfigurační soubor `/etc/hosts`

```
1 127.0.1.1       prestashop-1-7.loc
2 127.0.1.1       prestashop-1-6.loc
3 127.0.1.1       prestashop-1-5.loc
4 127.0.1.1       prestashop-1-4.loc
5 127.0.1.1       prestacure.loc
```

Po spuštění virtuálního stroje se automaticky spustí webový server i databáze. Není nutné nic dalšího instalovat. Stačí pouze spustit internetový prohlížeč a požadovanou webovou aplikaci (případně lze využít odkazů v sekci oblíbené):

- PrestaCure – `http://prestacure.loc`,
- PrestaShop 1.7.8.5 – `http://prestashop-1-7.loc`,
- PrestaShop 1.6.1.15 – `http://prestashop-1-6.loc`,
- PrestaShop 1.5.0.17 – `http://prestashop-1-5.loc`,
- PrestaShop 1.4.7 – `http://prestashop-1-4.loc`.

Pokud by uživatel chtěl editovat zdrojové soubory webových aplikací, nebo vytvářet nové penetrační testy v rámci aplikace PrestaCure, je k dispozici předinstalovaný textový editor Visual Studio Code [35]. Pro jednodušší práci s databází byl nainstalován nástroj TablePlus, zobrazující databázi v přehledném grafickém prostředí [36].

Kořenový adresář všech aplikací se nachází ve `/var/www/`. Výchozí zdrojové soubory lze najít v adresáři `/home/prestacure/Desktop/Diplomova_prace`, kde lze nalézt i soubor `hesla_postupy.txt` s nápovědou a dalšími hesly, například k databázi MySQL. Zdrojové soubory aplikace PrestaCure lze rovněž nalézt v příloze D.

5 Praktická implementace aplikace

Tato část práce se zabývá návrhem a implementací vlastní webové aplikace vytvořené pomocí PHP frameworku Laravel (viz kapitola 1.1.2). Součástí je i MySQL databáze pro uchovávání dat. Implementovaná aplikace byla vyvíjena jako vícestránková (viz kapitola 1.3) s asynchronními prvky pro načítání některých komponentů bez nutnosti obnovovat stránku v prohlížeči.

5.1 Popis fungování webové aplikace PrestaCure

Pro lepší pochopení technické struktury webové aplikace PrestaCure je vhodné nejprve představit její fungování z pohledu uživatele. Popis je doplněn referencemi na grafický návrh, který proběhl před samotnou implementací aplikace.

Prvním krokem je registrace/přihlášení do aplikace formulářem zveřejněným na úvodní stránce webové aplikace (viz příloha A.1). Uživatel vyplní přihlašovací údaje a je vpuštěn do řídícího panelu, neboli Dashboardu (viz příloha A.2), který je koncipován na akční panel na pravé straně a menu na straně levé. Jako první se zobrazí panel Projects, kde uživatel může zakládat projekty uvedením URL webové stránky se systémem PrestaShop či jiným, případně je i mazat.

Jako první položka v menu je zobrazena ikona se jménem uživatele. Po kliknutí na ni se objeví další dvě možnosti - Account a Log Out. Account (viz příloha A.3) zprostředkovává možnosti úpravy uživatelského profilu - v tomto případě změnu hesla. Možnost Log Out uživatele odhlásí z aplikace a přesměruje na úvodní stránku.

Každý nový projekt je při zakládání ověřován z pohledu vlastnictví webového hostingu uživatelem. Smyslem implementované aplikace není poskytnout nástroj, umožňující útočníkům získat informace o zranitelnostech cílených webových stránek, ale především varovat majitele před možnými útoky a zprostředkovat způsob zabezpečení vůči nalezeným zranitelnostem. Samotné ověření vlastnictví je realizováno požadavkem na umístění souboru `prestacure.html` do kořenového adresáře domény. Pokud zde takový soubor existuje, vlastnictví je ověřeno.

Následně je možné přejít do panelu konkretního projektu a spustit test. Po testu se zobrazí přehledná zpráva s výsledky v podobě tabulky, případně odkazy na možné opravy nalezených zranitelnosti (viz příloha A.4).

Dalšími položkami v menu webové aplikace jsou Admin Panel, Tests, Home a Help. Admin Panel (viz příloha A.5) je zobrazen pouze administrátorovi webové aplikace a umožňuje opětovné načtení všech dostupných penetračních testů do databáze. Tests (viz příloha A.6) obsahuje vysvětlivky pro veškeré penetrační testy, které aplikace PrestaCure poskytuje. Home odkazuje na domovskou stránku a Help na pomocnou ilustraci zobrazující postup pro provedení testování.

5.2 Architektura vlastní aplikace

Pro konkrétní představu o životním cyklu požadavku v rámci aplikace implementované ve frameworku Laravel byl vytvořen vývojový diagram v příloze B.1. Kód aplikace byl psán s ohledem na konvence využité architektury a minimalizuje replikaci řádků díky využívání znovupoužitelných komponent.

Architektura samotné aplikace využívá od podstaty frameworku Laravel modelu MVC (viz kapitola 1.2.1), podle kterého je definována i adresářová struktura, skládající se z 11 složek. V následujících kapitolách budou rozebrány ty, ve kterých byly prováděny změny a vytvářeny nové soubory. Adresářová struktura je také detailně popsána ve zdroji [37].

5.2.1 Adresář `/routes`

Adresář `/routes` obsahuje soubory `web.php` a `api.php` sloužící ke směrování požadavků ke konkrétním zdrojům aplikace. Jak je vidět ve výpisu 5.1, tyto soubory definují obsah odpovědi aplikace tím, že specifikují kontrolér, který požadavek zpracuje, a případně i tzv. middleware – mechanismus pro kontrolu a filtrování požadavků HTTP vstupujících do aplikace. Framework Laravel například poskytuje middleware `auth`, který kontroluje, jestli je klient přihlášený nebo ne a případně provádí adekvátní přesměrování. Byl implementován i vlastní middleware, kontrolující, zda je přihlášený uživatel administrátor. Pokud ano (jedná se o prvního registrovaného uživatele), je v menu uživateli zobrazena položka Admin Panel. Soubor `web.php` zprostředkovává cesty do uživatelsky přístupných koncových uzlů prostřednictvím webového prohlížeče. Soubor `api.php` zprostředkovává přístup k implementovanému REST API (více viz kapitola 5.5).

Výpis 5.1: Příklad záznamu cesty v souboru `web.php`

```
1 Route::get('/', HomeShowController::class)->name('home');
2
3 Route::put('/verify/{project:slug}', [ProjectController::
    class, 'update'])->middleware('auth')->name('verify-
    project');
4
5 Route::get('/dashboard', DashboardIndexController::class)->
    middleware('auth')->name('dashboard');
6
7 Route::post('/dashboard/add-project', [ProjectController::
    class, 'create'])->middleware('auth');
```

5.2.2 Adresář `/resources/views`

V adresáři `/views` lze nalézt speciální PHP soubory spravované šablonovacím systémem Blade (součást frameworku Laravel), který kompiluje všechny Blade šablony (soubory s příponou `.blade.php`) do prostého kódu PHP. V podstatě je zde vytvářen front-end webové aplikace. Kontroléry potom tyto šablony využívají při generování požadované stránky, a zároveň je plní aplikačními daty. Šablony jsou následně uloženy do mezipaměti, dokud nejsou upraveny. Soubory v tomto adresáři jsou rozděleny do dalších adresářů, obsahujících znovupoužitelné komponenty, které se využívají v dalších souborech `.blade.php`.

5.2.3 Adresář `/app`

Adresář `/app` obsahuje mimo jiné adresáře `/Http/Controllers`, definující kontroléry a `/Models`, kde jsou vytvářeny relační vztahy mezi jednotlivými databázovými tabulkami (více v kapitole 5.3).

Kontroléry v adresář `/Http/Controllers` definují veškerou logiku zpracování požadavků určitého druhu. Například pro zpracování požadavku na zobrazení všech testů v aplikaci byl vytvořen kontrolér `TestIndexController.php`:

Výpis 5.2: Kontrolér `TestIndexController.php`

```
1  namespace App\Http\Controllers;
2
3  use App\Models\Test;
4
5  class TestIndexController extends Controller
6  {
7      /**
8       * Index all available tests.
9       *
10      * @return view
11      */
12     public function __invoke()
13     {
14         return view('layouts.tests', [
15             'tests' => Test::all(),
16         ]);
17     }
18 }
```

Tento kontrolér interaguje s Blade šablonou `layouts.tests.blade.php` a poskytuje jí proměnnou `tests`, která obsahuje veškeré penetrační testy v aplikaci v rámci databázového modelu `Test` (více viz 5.3).

V adresáři `/app` se také nachází podadresář `/Tests`, ve kterém jsou umístěny definice implementovaných testů - více v kapitole 5.4.

5.3 Databáze

Pro správné fungování webové aplikace PrestaCure je potřeba databáze. V tomto projektu byla využita databáze MySQL. Její struktura je znázorněna v ERD diagramu (Entity-relationship model) v příloze C.1. Ve frameworku Laravel je databáze realizována dvěma součástmi – vytvořením relačního modelu a databázového schématu, tzv. migrace.

Schémata databázových tabulek se vytváří v adresáři `/database/migrations`. Jedná se o PHP soubory definující tabulkové sloupce, jak je uvedeno ve výpisu 5.3. Jsou v něm definovány sloupce `id`, `foreignId` a `timestamps`:

Výpis 5.3: Příklad schématu databázové tabulky

```
1 Schema::create('histories', function (Blueprint $table) {
2     $table->id();
3     $table->foreignId('project_id');
4     $table->timestamps();
5 });
```

Relační modely se vytváří ve zmíněném adresáři `/app/Models`. Model označuje databázovou tabulku. V každém modelu se definuje jeho vztah k jinému modelu:

Výpis 5.4: Příklad vytvořené relace mezi databázovými modely

```
1 class Result extends Model
2 {
3     public function tests()
4     {
5         return $this->hasOne(Test::class, 'id', 'test_id');
6     }
7 }
```

V příkladu výpisu 5.4 je definován relační vztah mezi objekty modelu `Result`, který je ve vztahu k právě jednomu objektu modelu `Test`. Tyto vztahy následně

umožňují využití ORM nástroje Eloquent (viz kapitola 1.1.2), který zprostředkovává jednodušší dotazování vztažených objektů v kódu aplikace například takto:

```
1 $test = Result::first()->tests;
```

Protože ORM Eloquent v odpovědi vrací datový typ `Collection`, je možné používat funkce k němu přiřazené, jako například `first()`, která vybere první element kolekce.

5.3.1 Načítání testů do databáze a vytváření nových

Laravel umožňuje při migraci databázových tabulek provádět jejich naplnění daty. Při inicializaci aplikace jsou tedy všechny penetrační testy automaticky načteny pomocí třídy `DatabaseSeeder` v souboru `database/seeders/DatabaseSeeder.php` a funkce `loadTests()` vytvořené v souboru `app/Tests/LoadAllTests.php`.

Aby bylo toto možné provést, musí být každý test reprezentován třídou implementující PHP rozhraní `/app/Tests/TestInterface.php` s definovanými statickými funkcemi, viz výpis 5.5. Jedná se o funkce získávající název, odkaz na možné řešení a popis penetračního testu. Poslední funkcí je funkce `detect()` s parametrem `Project $project`, který funkci předává instanci projektu uloženého v databázi. V případě nutnosti tak může být využito již získaných informací o projektu z předešlých testování. Tato funkce je volána při spouštění jakéhokoliv testu a měla by obsahovat hlavní logiku penetračního testu.

Výpis 5.5: PHP rozhraní `TestInterface`

```
1 interface TestInterface
2 {
3     public static function getName();
4     public static function getFixLink();
5     public static function getDescription();
6     public static function detect(Project $project);
7 }
```

Funkce `loadTests()` (zobrazená ve výpisu 5.6) tedy projde všechny soubory v adresáři `/app/Tests` s definovanými výjimkami, z každé extrahuje zmíněné proměnné a uloží je do databáze. Penetrační testy jsou tak připraveny k použití v aplikaci. Pozor, pokud je již aplikace spuštěná a zpřístupněná uživatelům, nesmí dojít ke změně názvu testu nebo jeho smazání. To by totiž mohlo vést k přegenerování databáze testů a změnám jejich ID, které by potom nekorelovaly s výsledky.

Výpis 5.6: Zkrácený výpis funkce `loadTests()`

```
1  public static function loadTests(): void
2  {
3      $classes = array_values(array_diff(scandir(__dir__),
           array('.', '..', 'TestInterface', 'LoadAllTests.php',
           'TestInterface.php', 'TestsHelperFunctions.php')));
4
5      foreach ($classes as $class) {
6          $class_name = strstr($class, '.', true);
7          $instance = '\App\Tests\\' . $class_name;
8          $test_type = new $instance;
9
10         Test::updateOrCreate(
11             ['class' => $class_name],
12             ['name' => $test_type::getName(),
13             'description' => $test_type::getDescription(),
14             'fix_link' => $test_type::getFixLink()
15             ]
16         );
17     }
18 }
```

Výsledky mohou být uživateli prezentovány několika způsoby. Pomocí REST API (viz 1.2.2) a v rámci frontendu webové aplikace PrestaCure. Aby bylo provádění penetračních testů pro uživatele aplikace co nejpřívětivější, je použita knihovna `jQuery` pro posílání asynchronních požadavků na server bez nutnosti obnovovat stránku v prohlížeči. Nové výsledky tak nejsou načítány přímo z databáze, ale přeposílány přímo. Zároveň jsou samozřejmě ukládány do databáze, odkud jsou při obnovení nebo znovu otevření Dashboardu minulé výsledky provedených testů načítány.

Všechny testy také musí vracet výsledky v určité struktuře a ve formátu JSON, jak je znázorněno ve výpisu 5.7. Jsou povinné čtyři proměnné (v závorce jsou specifikovány jejich datové typy):

- `(integer) test_id` – slouží pro získání ID testu z databáze,
- `(string) test_name` – slouží pro získání názvu testu z PHP třídy,
- `(string) info` – označuje popis výsledku penetračního testu,
- `(boolean) vulnerable` – hodnotou `true` označuje testovanou aplikaci zranitelnou a naopak,
- `(string) fix_link` – slouží pro získání odkazu na potenciální odstranění zranitelnosti z PHP třídy (pokud byl test neúspěšný, není tato hodnota povinná).

Výpis 5.7: Povinná struktura návratové hodnoty testů

```
return json_encode([
    'test_id' => Test::where('name', self::getName())->first
        ()->id,
    'test_name' => self::getName(),
    'info' => 'Test result description',
    'vulnerable' => true,
    'fix_link' => self::getFixLink()
]);
```

5.4 Penetrační testy v aplikaci

Jak již bylo zmíněno v kapitole 5.2, veškeré testy využívané pro testovaní aplikací specifikovaných administrátorem aplikace PrestaCure jsou definovány v adresáři `/app/Tests`. V následujících kapitolách jsou popsány jednotlivé penetrační testy implementované v aplikaci PrestaCure.

5.4.1 Ověření verze PrestaShopu

Základním testem webové aplikace PrestaCure je ověření skutečnosti, že testovaná aplikace běží na systému PrestaShop a v jaké verzi. Pokud ovšem ale aplikace není vyhodnocena jako PrestaShop, další testy jsou i přes to provedeny. Primárně slouží výsledky tohoto testu pro využití v dalších penetračních testech, jejichž zkoumané zranitelnosti jsou například zaměřené pouze na určitou verzi systému PrestaShop.

Funguje na principu vyhledávání určitých signatur v kódu HTML a postupným dotazováním na specifické zdrojové souborů a HTTP hlavičky. Test je definován v souboru `/app/Tests/PrestaShopVersion.php`, kde lze rovněž nalézt zmíněné signatury. Pokud jsou nalezeny alespoň tři specifikované signatury, je webová aplikace označena jako PrestaShop verze 1.7, 1.6, 1.5, nebo 1.4.

Výsledku tohoto testu samozřejmě nejsou naprosto jednoznačné, neboť se mohou vyskytnout případy, kdy se vývojáři snaží zmást podobné nástroje vystavováním falešných signatur. Nicméně, v případě zkoumaných verzí systému PrestaShop byly bezchybné.

5.4.2 Detekce přítomnosti souboru `.git`

Tento test prohledá kořenový adresář webové aplikace a vyhledává soubor `.git`, který vývojáři často zapomínají odstranit, nebo omylem nahráli na webový server.

Tato složka obsahuje součásti zdrojového kódu uchovávané v rámci verzovacího systému Git – nástroji, který sleduje všechny úpravy souborů nebo složek v projektech a umožňuje vývojářům se v případě potřeby vracet k minulým řešením [38].

Pokud je soubor .git ponechán na webovém serveru jako veřejně dostupný, tak se útočníkovi otevírá možnost obnovit a prozkoumat zdrojový kód aplikace. Může tak nalézt potenciální zranitelnosti, zjistit, jak obejít implementovaná bezpečnostní opatření nebo zneužít některá důvěrná data [38].

V případě, že je soubor `.git` zveřejněný, výsledek testu informuje uživatele o této skutečnosti a doporučí soubor odstranit.

5.4.3 Kontrola bezpečnostních hlaviček

Bezpečnostní hlavičky nastavené v odpovědích webového serveru jsou často opomíjeným způsobem aktivní ochrany webových aplikací, které může zabezpečit moderní internetové prohlížeče před snadno mitigovatelnými zranitelnostmi a útoky typu XSS a clickjacking. Tento test v rámci aplikace PrestaCure ověřuje, zda jsou webové stránky zabezpečeny pomocí těchto bezpečnostních hlaviček: [39]

- **HSTS** – hlavička Strict Transport Security je určena pro konfiguraci prohlížečů tak, aby komunikovaly pouze a jen prostřednictvím zabezpečeného protokolu (používá se k ochraně před útoky typu „man-in-the-middle“ vynucením veškeré komunikace na HTTPS protokolu),
- **X-Frame-Options** – zaručuje ochranu webových aplikací proti clickjackingu (obsah stránek není povoleno zobrazovat na jiných stránkách při použití HTML elementů <frame>, <iframe> nebo <object>),
- **X-Content-Type-Options** – zabraňuje prohlížeči interpretovat soubory jako jiný MIME typ, než jaký je uveden v hlavičce Content-Type HTTP,
- **Referrer-Policy** – řídí, které informace jsou odesílány v HTTP hlavičce Referer,
- **Content-Security-Policy** – zabraňuje široké škále útoků (včetně XSS) tak, že omezuje načítání zdrojů webové stránky z povolených URL,
- **Permissions-Policy** – umožňuje webům přísněji omezit, kterým entitám lze udělit přístup k jejich funkcím.

Pokud je zjištěno, že některá ze zkoumaných bezpečnostních hlaviček není nastavena, je na to uživatel upozorněn a odkázán na webové stránky s návodem pro jejich nastavení.

5.4.4 CVE-2018-19355

Bezpečnostní chyba zařazená v databázi zranitelností CVE (Common Vulnerabilities and Exposures), která se vyskytuje v doplňku Customer Files Upload pro systém PrestaShop verze 1.5 až 1.7, umožňuje útočníkům spustit libovolný kód nahráním PHP souboru přes `modules/orderfiles/upload.php` s hodnotou auptype rovnou `product` (pro cíle nahrávání v `modules/productfiles`), `order` (pro cíle nahrávání v `modules/files`) nebo `cart` (pro cíle nahrávání v `modules/cartfiles`) [40].

Penetrační test této zranitelnosti tedy nejdříve ověřuje verzi PrestaShopu a případně zobrazí varování. Řešením této chyby je aktualizace modulu nebo odstranění či oprava zranitelných souborů.

5.4.5 Spamový útok kontaktního formuláře

V roce 2021 proběhl poměrně masivní útok na výchozí kontaktní formulář, který je používán ve většině funkčních implementacích PrestaShopu. Tento formulář slouží zákazníkům v případě potřeby pro kontaktování zákaznického servisu, nebo administrátora stránek. Avšak minimálně v PrestaShop verzích 1.6.0.9–1.6.1.16 umožňoval při vytvoření jednoduchého HTTP POST požadavku odesílat neomezené množství dotazů:

Výpis 5.8: HTTP POST požadavek vytvářející spam

```
1 <prestashop_url>/?controller=contact&submitMessage=1&message=
    SpamZprava&from=test@test.cz&id_contact=1
```

To se projeví zahlcením schránky s těmito zprávami v administraci PrestaShopu, případně i emailové schránky správce. Ve verzi 1.6.1.17 byla tato zranitelnost již opravena díky využití jedinečných generovaných CSRF tokenů, používaných proti útokům Cross Site Request Forgery (viz [41]), umístěných jako skrytý element v HTML kódu formuláře. Tyto tokeny je následně nutné odesílat spolu s formulářem pro ověření legitimního uživatele. Takovéto řešení znemožňuje odesílat výše zmíněné HTTP POST požadavky.

Z pohledu testování je situace problematická. Zmíněné řešení zkoumané zranitelnosti je implementováno tak, že i uměle vytvořeným požadavkům na formulář vrací v odpovědi HTTP status kód 200 – i když jsou odeslané formuláře na aplikační úrovni zablokovány. Útočník tak nemá možnost jednoduše otestovat, zda požadavek proběhl v pořádku či nikoli.

Penetrační test je tedy zkonstruován spíše jako doporučení, protože možnou zranitelnost nelze jednoznačně prokázat. Ověřuje, zda webová aplikace běží na uvedených zranitelných verzích systému PrestaShop, a jestli je při odesílání formuláře používán zmíněný token. Následně uživatele upozorní, pokud by jeho aplikace mohla být zranitelná a doporučí zabezpečení.

5.5 REST API

V rámci webové aplikace je spuštěno i REST API (viz kapitola 1.2.2), které zprostředkovává ty stejné funkcionality jako samotná webová aplikace. REST API v odpovědi ze své definice odesílá reprezentaci požadovaného zdroje ve formátu JSON.

Díky implementovanému autorizačnímu systému, popsanému v kapitole 5.5.1, zaručuje vrácení výsledků náležící pouze přihlášenému uživateli. Je také správně navrženo z hlediska jeho verzování, kdy všem koncovým bodům předchází prefix `api/v1/`. V případě vytvoření nové verze API by tak to starší zůstalo zachováno pro dodržení kompatibility a pro jeho novou verzi by byl vytvořen nový prefix `api/v2/`.

REST API jsou definováno v souboru `routes/api.php`, ze kterého jsou volány jednotlivé kontroléry obsahující přehledně definované RESTful funkce jako:

- `index` – výpis veškerých zdrojů,
- `create` – vytvoření nových zdrojů,
- `show` – výpis informací o konkrétním zdroji,
- `update` – aktualizace zdrojů,
- `delete` – smazání zdrojů.

Zmíněné kontroléry jsou definovány v souborech (s prefixem `RestApi/v1/`):

- `AuthController.php` – pro vykonávání autentizačních a autorizačních kroků,
- `AllTestsController.php` – pro spouštění všech testů,
- `TestController.php` – pro spouštění konkrétního testu,
- `ProjectHandleController.php` – pro obsluhování požadavků týkajících se konkrétního projektu,
- `ProjectsController.php` – pro obsluhování požadavků týkajících se projektů,
- `ProjectResultsController.php` – pro zobrazování výsledků provedených testů projektu.

Díky svým možnostem může uživatel aplikovat celou řadu automatizačních procesů. Je vhodné také zmínit, že API využívá společné funkce se samotnou webovou aplikací v nejvyšší možné míře a nedochází tak k duplikaci kódu.

5.5.1 Zabezpečení REST API

Všechny koncové body REST API jsou zabezpečeny díky autorizaci realizované pomocí tzv. bearer tokenů generovaných při přihlášení uživatele na veřejně dostupném koncovém bodu `/login` v rámci standardizovaného frameworku OAuth 2.0 [42].

Bearer token je v podstatě řetězec znaků určité délky, poskytující dočasný přístup k systému. Přihlašovací rozhraní je dostupné pomocí HTTP metody POST, při které je nutné v těle požadavku specifikovat hodnoty `e-mail` a `heslo` již existujícího uživatele zaregistrovaného v aplikaci PrestaCure. REST API na tento požadavek odpoví zprávou s autorizačním bearer tokenem. Pro lepší představení komunikace při přihlašování do REST API byl vytvořen vývojový digram v příloze B.2. Registrace nového uživatele do aplikace přes API není z bezpečnostních důvodů povolena.

Vytvořený token je nutné posílat v každém dalším požadavku, jinak dojde k jeho odmítnutí. K odhlášení lze použít koncový bod `/logout`, který po úspěšném provedení vlastní funkce smaže autentizační token z databáze a znemožní tak jeho další přístup k funkcionalitám aplikace.

Ve frameworku Laravel je k implementaci podobného autentizačního systému možné použít rozšíření Laravel Sanctum, které pomocí middlewaru `auth:sanctum` usnadňuje práci s generováním a zpracováváním autorizačních tokenů v aplikaci. Z těchto důvodů bylo rozšíření použito i při implementaci aplikace PrestaCure.

Zkrácený výpis 5.9 zobrazuje příklad přihlašovacího požadavku na REST API se zadaným e-mailem a heslem. Příkladem odpovědí API na tento požadavek je zpráva ve formátu JSON (pokud jsou přihlašovací údaje správné) znázorněná ve výpisu 5.10.

Výpis 5.9: HTTP požadavek pro přihlášení do REST API

```
1  POST /api/v1/login HTTP/1.1
2  Content-Type: multipart/form-data; boundary=---XXX
3  Content-Length: 293
4  ---XXX
5  Content-Disposition: form-data; name="email"
6  <uživatelský e-mail>
7  ---XXX
8  Content-Disposition: form-data; name="password"
9  <heslo>
10 ---XXX--
```

Výpis 5.10: Odpověď na HTTP požadavek pro přihlášení do REST API

```
{
    "message": "Welcome in PrestaCure API",
    "user": {
        "id": <id>,
        "username": <uživatelské jméno>,
        "email": <uživatelský e-mail>,
        "created_at": <časové razítko>,
        "updated_at":  <časové razítko>
    },
    "token": "<autorizační bearer token>"
}
```

5.5.2 Koncové body REST API

Jak bylo zmíněno v minulé kapitole, při komunikaci s REST API je nutné se nejdříve přihlásit a používat vygenerovaný token obsažený v každém požadavku. Takovýto požadavek může vypadat například jako ve výpisu 5.11. Naopak příkladem odpovědi na tento požadavek je výpis 5.12.

Výpis 5.11: Příklad požadavku s autorizačním tokenem

```
GET /api/v1/test/1 HTTP/1.1
Accept: application/json
Authorization: Bearer <autorizační token>
Host: prestacure.loc
```

Výpis 5.12: Příklad struktury JSON odpovědi

```
{
    "id": <id>,
    "name": <název testu>,
    "description": <popis testu>,
    "fix_link": <odkaz na možný postup opravy>,
    "created_at": <časové razítko>,
    "updated_at": <časové razítko>
}
```

K dispozici jsou koncové body – identifikátory URL (viz tabulka 5.1), kde jsou všechny pojmenovány se zmíněným prefixem `/api/v1`. U všech je také zvýrazněna použitá HTTP metoda. Je také důležité zmínit, že pokud byl požadavek vyhodnocen jako chybný, je uživateli vrácena informační zpráva se status kódem 400.

Tab. 5.1: Tabulka koncových bodů REST API

HTTP metoda	**Koncový bod**	**Popis**
GET	`/test/<test_id>`	Vrátí informace o konkrétním testu specifikovaném pomocí parametru `test_id` v URL požadavku.
POST	`/test/run/specific`	Je spuštěn pouze test se specifikovanými hodnotami `project_id` a `test_id` v těle požadavku a vrácen výsledný report.
GET	`/tests`	Vrátí kompletní seznam všech implementovaných penetračních testů, včetně jejich ID atd.
GET	`/tests/search/<str>`	Vyhledá a zobrazí testy podle specifikovaného řetězce v URL požadavku.
POST	`/tests/run/all`	Spustí všechny definované testy pro specifikovaný projekt pomocí `project_id` v těle požadavku a vrátí výsledný report.
GET	`/project/<project_id>`	Vrátí informace o projektu specifikovaného pomocí parametru `projet_id` v URL parametru požadavku.
DELETE	`/project/<project_id>/delete`	Smaže projekt specifikovaný pomocí parametru `projet_id` v těle požadavku.

GET	`/project/<project_id>/results`	Vrátí výsledky všech provedených testů projektu specifikovaného pomocí parametru `projet_id` v URL parametru požadavku.
PUT	`/project/<project_id>/verify`	Ověří, zda je v rámci specifikované domény projektu v URL parametru přítomný soubor `prestacure.html` (viz kap 5.1).
GET	`/projects`	Vrátí seznam všech uživatelem založených projektů.
GET	`/projects/search/<str>`	Vyhledá všechny projekty podle specifikovaného řetězce v URL požadavku.
POST	`/projects/addproject`	Přidá projekt podle domény zadané parametrem `new_project` v těle požadavku, u kterého je dále nutné ověřit jeho vlastnictví (viz kap 5.1).
POST	`/login`	Slouží k přihlášení do REST API. Více viz kapitola 5.5.1.
POST	`/logout`	Slouží k odhlášení z REST API.

5.6 Instalace aplikace PrestaCure

Projekty frameworku Laravel mají ve své adresářové struktuře zapsány citlivé informace důležité pro bezpečnost aplikace. Jedná se zejména o přístupové údaje k databázi a tzv. aplikační klíč APP_KEY, který se používá k šifrovacím procesům. Všechna data, která jsou v aplikaci šifrována, derivují další klíče právě z klíče APP_KEY. Proto je nutné projekty distribuovat bez těchto citlivých informací, které jsou uchovávány v souboru `.env`.

K minimalizaci přenosu souborů nutných ke spuštění projektu jsou v kořenovém adresáři frameworku Laravel v souboru `composer.lock` zaznamenány veškeré knihovny a jejich verze, které je nutné doinstalovat při instalaci aplikace. Ve stejném adresáři se nachází i soubor `composer.json`, který obsahuje informace o nejaktuálnějších verzích využitých knihoven. Při aktualizaci projektu jsou přepsány zastaralé verze knihoven v souboru `composer.lock` a dojde i k aktualizaci knihoven.

V případě PHP a Laravelu je tedy doporučováno využití nástroje Composer (viz kapitola 4.4). Zároveň je požadována instalace PHP verze 8.0. Postup samotné instalace aplikace je následující:

1. Příkaz `composer install` – nainstaluje veškeré knihovny a součásti ze souboru `composer.lock`.
2. Přejmenování souboru `.env.example` na `.env` a vepsání přihlašovacích údajů k databázi.
3. Příkaz `php artisan key:generate` – vygenerování nového APP_KEY.
4. Příkaz `php artisan migrate:fresh --seed` – obnovení a naplnění databáze výchozími daty.

Následně je možné aplikaci spustit příkazem `php artisan serve`, nebo použitím jakéhokoliv webového serveru s kořenovým adresářem v `/public`. Výchozí přihlašovací údaje jsou zaznamenány ve výpisu 4.3. Zdrojové soubory aplikace PrestaCure lze nalézt v příloze D.

Závěr

Cílem teoretické části této diplomové práce bylo se seznámit s aktuálním vývojem v oblasti penetračního testování e-commerce platformy PrestaShop, s frameworkem Laravel/Nette a REST API. Cílem praktické části práce bylo ve vybraném frameworku implementovat webovou aplikaci schopnou provádět penetrační testy pro platformu PrestaShop.

V teoretické části práce byly představeny technologie využité pro implementaci webové aplikace jako PHP, MySQL a framework Laravel. Byla popsána i architektura Model-View-Controller, sloužící pro přehledný návrh a organizaci kódu, a stejně tak i REST. V dalších kapitolách bylo popsáno penetrační testování a rozebrán systém PrestaShop s ohledem na jeho architekturu a bezpečnost. Vzhledem ke zjištění, že se na trhu prakticky nenachází žádný automatizovaný produkt pro penetrační testování webových aplikací stavěných na platformě PrestaShop, bylo přistoupeno k implementaci aplikace PrestaCure.

V rámci praktické části práce byla ve frameworku Laravel vytvořena vlastní webová aplikace PrestaCure umožňující uživatelům otestovat si jejich webové aplikace implementovanými penetračními testy. Po přihlášení je uživatel přesměrován do Dashboardu, kde může vytvářet nové projekty a v každém z nich spouštět testy buď po jednom, nebo všechny zároveň. Obsahuje také sekci s vysvětlením jednotlivých testů. Tato sekce se spolu s testy zobrazenými u projektů generuje dynamicky podle dat uložených v databázi. Administrátorovi je také zpřístupněna možnost načtení nových penetračních testů stisknutím jediného tlačítka. Aplikace byla navržena tak, aby vyhovovala moderním standardům a byla uživatelsky přívětivá. Její architektura navíc usnadňuje implementaci dalších penetračních testů.

Byla sestavena sada pěti penetračních testů. První test se zabývá ověřením skutečnosti, zda testovaná webová aplikace běží na systému PrestaShop a určuje také jeho verzi. Druhý test detekuje přítomnost souboru `.git` v kořenovém adresáři webové aplikace. Třetí test kontroluje nastavení bezpečnostních hlaviček zvyšujících zabezpečení vůči nejběžnějším formám útoků. Čtvrtý test byl vytvořen pro odhalení zranitelnosti CVE-2018-19355, která umožňuje provádět útoky RCE. Jako poslední byl implementován test upozorňující na možnost spamového útoku na kontaktní formulář aplikací PrestaShop. Výstupem penetračních testů je přehledná tabulka, která poskytuje informace o výsledcích testů s odkaz na mitigaci nalezených zranitelností. Implementovaná sada penetračních testů tedy úspěšně prokázala funkčnost webové aplikace PrestaCure. Souběžně s aplikací bylo vyvinuto i zabezpečené REST API poskytující stejné zdroje jako webová aplikace.

Všechny penetrační testy samozřejmě nejsou zaměřeny pouze na redakční systém PrestaShop. Je potřeba si ale uvědomit, že ty nejběžnější kybernetické útoky

na webové aplikace jsou platné v podstatě na jakýkoliv redakční systém, a proto je potřeba prověřit i je. Existence platformy pro testování pouze určitého systému ale zprostředkovává mnohem větší spolehlivost testů a důraz na specifické zranitelnosti.

Pro zjednodušení testování a další vývoj bylo připraveno i testovací prostředí ve formě virtuálního stroje, který obsahuje nakonfigurovaný webový server s aplikací PrestaCure a čtyřmi verzemi aplikací se systémem PrestaShop. Vývojáři dalších penetračních testů tak nemusí řešit nedostatek testovatelných subjektů a přípravu vývojového prostředí.

Závěrem lze konstatovat, že díky funkčnosti aplikace a implementovaným penetračním testům splňuje tato práce svůj účel. Díky absenci podobných nástrojů na trhu navíc poskytuje nové řešení pro majitele e-shopů se systémem PrestaShop, kteří si díky aplikaci PrestaCure mohou otestovat, zda je jejich obchod bezpečný jak pro ně, tak pro uživatele a v případě problémů chyby opravit díky poskytnutým návodům.

Samotná aplikace ovšem z druhé strany nese bezpečnostní riziko v případě, kdyby se dostala do rukou útočníka, který by tak mohl efektivně skenovat webové aplikace a zjišťovat, na co jsou zranitelné. Následně by se proces útoku a kompromitace výrazně zjednodušil. Proto implementovaná aplikace obsahuje procesy, které zamezí testování cizí webové aplikace.

Cílem dalšího vývoje aplikace PrestaCure by mohlo být rozšíření palety penetračních testů a možností webové aplikace. V průběhu implementace vlastní aplikace také došlo k publikaci nové verze frameworku Laravel, nabízí se tedy možnost aktualizace.

Literatura

[1] LE, Jessie. Prestashop Review 2021: You Definitely Can't Miss! *LitExtension* [online]. Tokyo: LitExtension, 2021 [cit. 2021-11-27]. Dostupné z URL: <https://litextension.com/blog/prestashop-review/>.

[2] LAW, Thomas J. 19 POWERFUL ECOMMERCE STATISTICS THAT WILL GUIDE YOUR STRATEGY IN 2021. *Oberlo* [online]. Vilnius: Oberlo, 2021 [cit. 2021-12-04]. Dostupné z URL: <https://www.oberlo.com/blog/ecommerce-statistics>.

[3] SKWARCZEK, Bartosz. As E-Commerce Booms, We Cannot Lose Sight Of Security. *Forbes* [online]. New York: Forbes, 2021 [cit. 2021-12-04]. Dostupné z URL: <https://www.forbes.com/sites/forbestechcouncil/2021/05/05/as-e-commerce-booms-we-cannot-lose-sight-of-security/?sh=63056ca25398>.

[4] Understand web applications. *Adobe* [online]. San José: Adobe, 2021 [cit. 2021-12-04]. Dostupné z URL: <https://helpx.adobe.com/ca/dreamweaver/using/web-applications.html>.

[5] Usage statistics of server-side programming languages for websites. *W3Techs* [online]. Maria Enzersdorf: Q-Success, 2021 [cit. 2021-12-04]. Dostupné z URL: <https://w3techs.com/technologies/overview/programming_language>.

[6] KOLADE, Chris. What is PHP? The PHP Programming Language Meaning Explained. *FreeCodeCamp* [online]. San Francisco: freeCodeCamp, 2021 [cit. 2021-12-04]. Dostupné z URL: <https://www.freecodecamp.org/news/what-is-php-the-php-programming-language-meaning-explained/>.

[7] What is MySQL? Everything You Need to Know. *Talend* [online]. Redwood City: Talend, [2021] [cit. 2021-12-04]. Dostupné z URL: <https://www.talend.com/resources/what-is-mysql/>.

[8] BROTHERTON, Claire. The Most Popular PHP Frameworks to Use in 2021. *Kinsta* [online]. West Hollywood: Kinsta, 2021 [cit. 2021-12-04]. Dostupné z URL: <https://kinsta.com/blog/php-frameworks/>.

[9] Laravel - The PHP Framework For Web Artisans. *Laravel* [online]. Arkansas, 2021 [cit. 2021-12-04]. Dostupné z URL: <https://laravel.com>.

[10] MIANG, Lia. Understanding Object-Relational Mapping: Pros, Cons, and Types. *AltexSoft* [online]. Foster City: AltexSoft, 2021 [cit. 2021-12-04]. Dostupné z URL: <https://www.altexsoft.com/blog/object-relational-mapping/>.

[11] BERNARD, Borek. Úvod do architektury MVC. *Zdroják.cz* [online]. Praha: Devel.cz Labs, 2009 [cit. 2021-12-04]. Dostupné z URL: <https://zdrojak.cz/clanky/uvod-do-architektury-mvc/>.

[12] MVC: Model, View, Controller. *Codecademy* [online]. New York: Ryzac, [2020] [cit. 2021-12-04]. Dostupné z URL: <https://www.codecademy.com/articles/mvc>.

[13] FIELDING, Roy Thomas. *Architectural Styles and the Design of Network-based Software Architectures* [online]. Irvine, 2000 [cit. 2019-11-08]. Dostupné z URL: <https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm>.

[14] BESTAIEVA, Diana. Web Applications Architectures: Components, Layers, and Types. *Cleveroad* [online]. Redwood City: Cleveroad, 2021 [cit. 2021-12-04]. Dostupné z URL: <https://www.cleveroad.com/blog/web-application-architecture>.

[15] PrestaShop Usage Statistics. *BuiltWith* [online]. Sydney: BuiltWith®, 2021 [cit. 2021-12-05]. Dostupné z URL: <https://trends.builtwith.com/shop/PrestaShop>.

[16] Create and build your online store with PrestaShop. *PrestaShop* [online]. Paris, France: PrestaShop, 2007 [cit. 2021-9-14]. Dostupné z URL: <https://www.prestashop.com>.

[17] Jakou verzi Prestashopu vybrat?. *OpenServis* [online]. Pardubice: OpenServis, 2020 [cit. 2022-05-11]. Dostupné z URL: <https://www.openservis.cz/prestashop-blog/jakou-verzi-prestashopu-zvolit/>.

[18] *Symfony* [online]. Clichy: Symfony SAS, 2022 [cit. 2022-05-11]. Dostupné z URL: <https://symfony.com>.

[19] PRESTASHOP 1.6 VS 1.7 - WHY YOU SHOULD UPGRADE TO THE LATEST VERSION OF PRESTASHOP?. *PrestaHERO* [online]. Thai Nguyen: PrestaHero, 2018 [cit. 2022-05-11]. Dostupné z URL: <https://prestahero.com/blog/post/15-upgrade-to-the-latest-version-of-prestashop.html>.

[20] PrestaShop - Vulnerability Trends Over Time. *CVE Details* [online]. Bedford: MITRE Corporation, 2021 [cit. 2021-12-05]. Dostupné z URL: <https://www.cvedetails.com/product/15797/Prestashop-Prestashop.html?vendor_id=8950>.

[21] S., Kirsten. Cross Site Scripting (XSS). *OWASP Foundation | Open Source Foundation for Application Security* [online]. Maryland: OWASP, [2021] [cit. 2021-12-05]. Dostupné z URL: <https://owasp.org/www-community/attacks/xss/>.

[22] SQL Injection. *OWASP Foundation | Open Source Foundation for Application Security* [online]. Maryland: OWASP, [2021] [cit. 2021-12-05]. Dostupné z URL: <https://owasp.org/www-community/attacks/SQL_Injection>.

[23] The Concept Of RCE (Remote Code Execution) Attack. *Wallarm* [online]. San Francisco: Wallarm, [2021] [cit. 2021-12-05]. Dostupné z URL: <https://www.wallarm.com/what/the-concept-of-rce-remote-code-execution-attack>.

[24] CVE-2018-8823. *CVE Details* [online]. Bedford: MITRE Corporation, 2018 [cit. 2021-12-05]. Dostupné z URL: <https://www.cvedetails.com/cve/CVE-2018-8823/>.

[25] Prepared Statements. *PHP: Hypertext Preproocessor* [online]. PHP: Hypertext Preproocessor [cit. 2022-05-11]. Dostupné z URL: <https://www.php.net/manual/en/mysqli.quickstart.prepared-statements.php>.

[26] Astra - PrestaShop Firewall. *Astra* [online]. Tucson: ASTRA IT, 2021 [cit. 2021-12-05]. Dostupné z URL: <https://www.getastra.com/prestashop-firewall>.

[27] How to secure a Prestashop ecommerce website? *HTTPCS* [online]. Montpellier: Ziwit SAS, [2021] [cit. 2021-12-05]. Dostupné z URL: <https://www.httpcs.com/en/security-plugin/prestashop>.

[28] MARTINÁSEK, Zdeněk. *Penetrační testování: Bezpečnost ICT2* [online prezentace]. Vysoké učení technické v Brně, 2018 [cit. 2021-12-05].

[29] CVE Details - The ultimate security vulnerability datasource. *CVE Details* [online]. Bedford: MITRE Corporation, 2021 [cit. 2021-12-05]. Dostupné z URL: <https://www.cvedetails.com/>.

[30] *OWASP Foundation | Open Source Foundation for Application Security* [online]. Maryland: OWASP, 2021 [cit. 2021-12-05]. Dostupné z URL: <https://owasp.org/>.

[31] OWASP Top Ten. *OWASP Foundation | Open Source Foundation for Application Security* [online]. Maryland: OWASP, 2021 [cit. 2021-12-05]. Dostupné z URL: <https://owasp.org/www-project-top-ten/>.

[32] *MAMP* [online]. Wörth am Rhein: MAMP, 2021 [cit. 2021-12-05]. Dostupné z URL: <https://www.mamp.info/>.

[33] Composer - A Dependency Manager for PHP. *Composer* [online]. Nils Adermann, 2012 [cit. 2021-12-05]. Dostupné z URL: <https://getcomposer.org>.

[34] FastCGI Process Manager (FPM). *PHP: Hypertext Preproocessor* [online]. PHP: Hypertext Preproocessor [cit. 2022-05-11]. Dostupné z URL: <https://www.php.net/manual/en/install.fpm.php>.

[35] *Visual Studio Code - Code Editing. Redefined* [online]. Seattle: Microsoft Corporation, 2015 [cit. 2022-05-14]. Dostupné z URL: <https://code.visualstudio.com>.

[36] *TablePlus | Modern, Native Tool for Database Management* [online]. Halifax NS: TablePlus, 2018 [cit. 2022-05-14]. Dostupné z URL: <https://tableplus.com>.

[37] Directory Structure. *Laravel* [online]. Arkansas: Laravel, [2021] [cit. 2022-05-11]. Dostupné z URL: <https://laravel.com/docs/8.x/structure>.

[38] What is GIT Source Code Exposure Vulnerability and Why Should You Care?. *IoSENTRIX* [online]. Herndon: ioSENTRIX, 2020 [cit. 2022-05-11]. Dostupné z URL: <https://iosentrix.com/blog/git-source-code-disclosure-vulnerability/>.

[39] OWASP Secure Headers Project. *OWASP Foundation | Open Source Foundation for Application Security* [online]. Maryland, [2021] [cit. 2022-05-11]. Dostupné z URL: <https://owasp.org/www-project-secure-headers/>.

[40] CVE-2018-19355 Detail. *National Institute of Standards and Technology* [online]. Maryland: NIST, 2018 [cit. 2022-05-11]. Dostupné z URL: <https://nvd.nist.gov/vuln/detail/CVE-2018-19355>.

[41] Cross Site Request Forgery (CSRF). *OWASP Foundation | Open Source Foundation for Application Security* [online]. Maryland: OWASP [cit. 2022-05-13]. Dostupné z URL: <https://owasp.org/www-community/attacks/csrf>.

[42] HARDT, D. *The OAuth 2.0 Authorization Framework* [online]. ISSN 2070-1721. Uděleno 2012. Dostupné z URL: <https://datatracker.ietf.org/doc/html/rfc6749>.

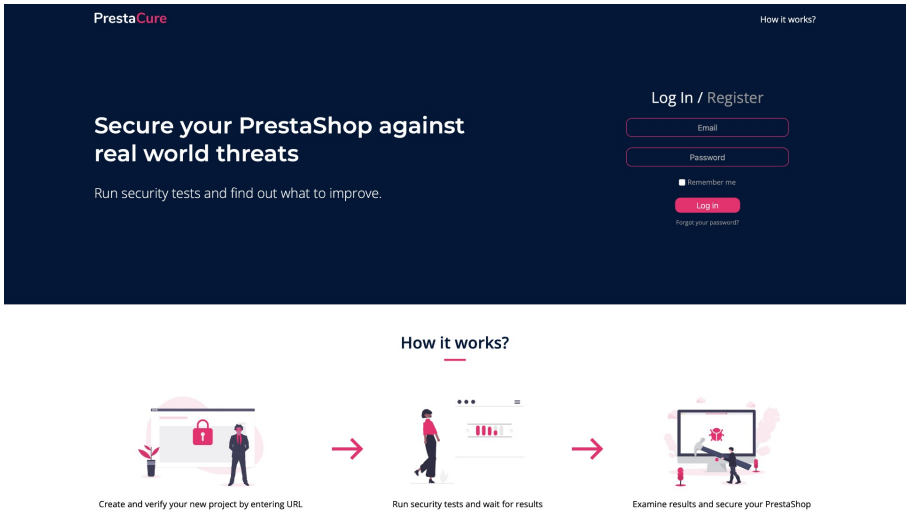
Seznam symbolů a zkratek

CMS	Content Management System
CSS	Cascading Style Sheets
ERD	Entity-relationship model
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
MVC	Model-View-Controller
ORM	Object Relational Mapper
OWASP	The Open Web Application Security Project
PHP	Hypertext Preprocessor
REST	Representational state transfer
SQL	Structured Query Language

Seznam příloh

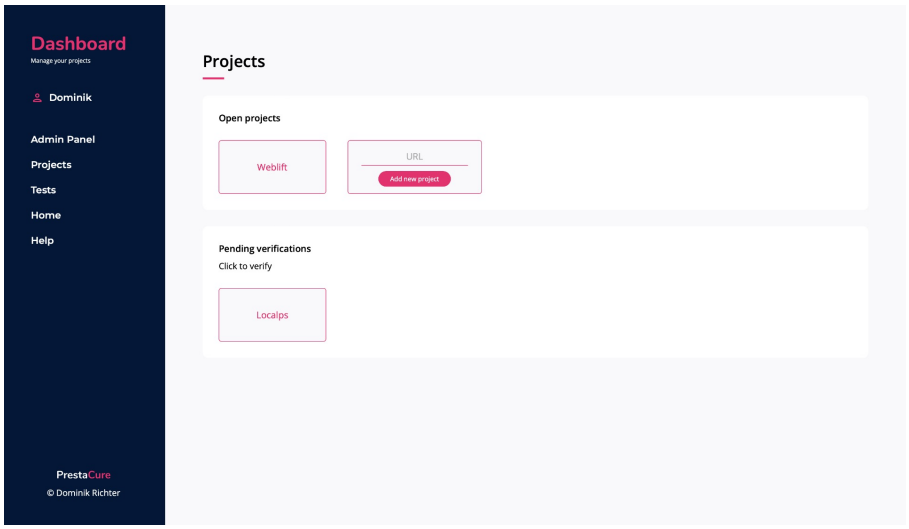
A Grafické návrhy webové aplikace

A.1 Domovská stránka



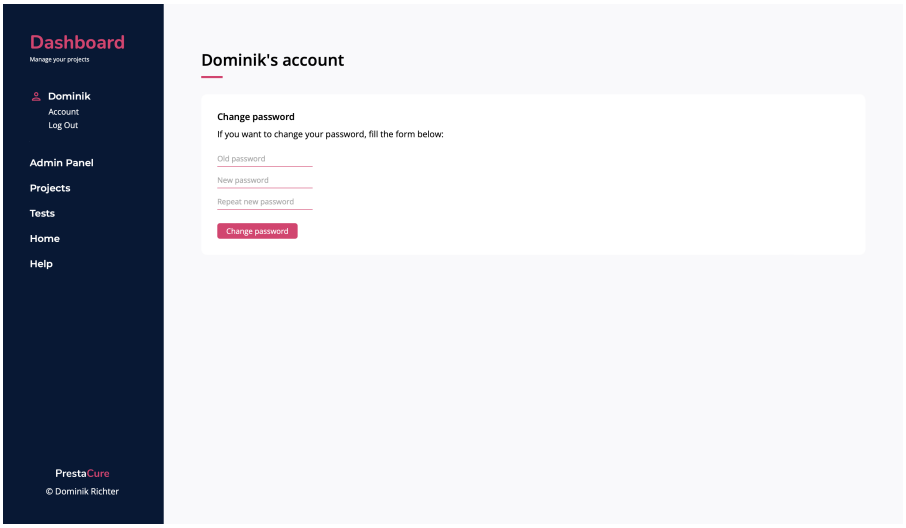
Obr. A.1: Grafický návrh domovské stránky

A.2 Dashboard



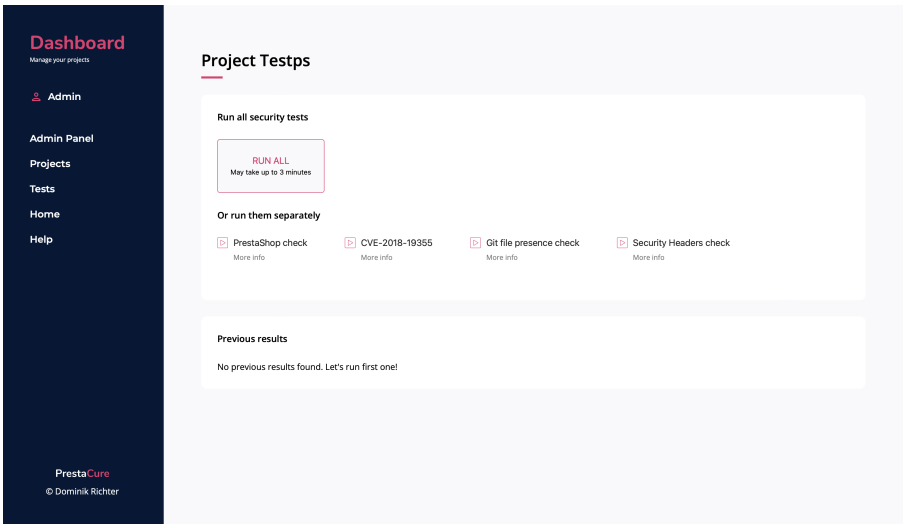
Obr. A.2: Grafický návrh dashboardu

A.3 Stránka uživatelského účtu



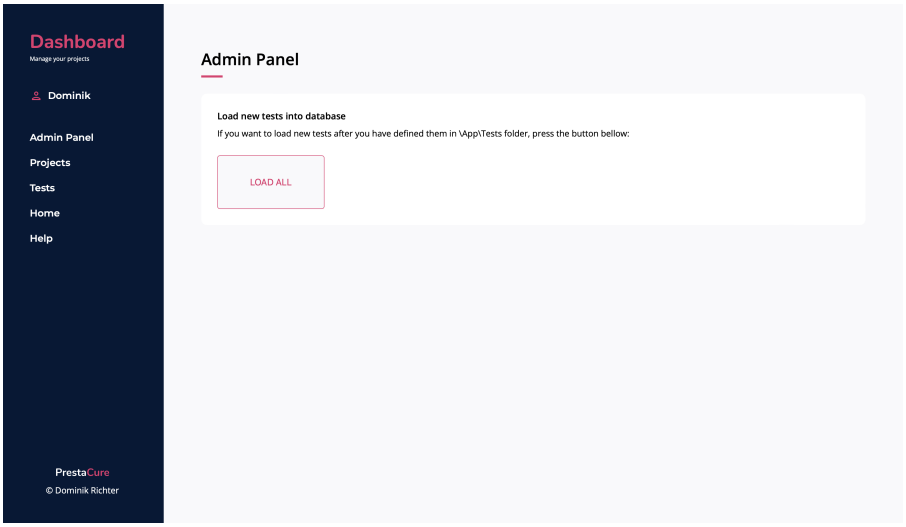
Obr. A.3: Grafický návrh stránky uživatelského účtu

A.4 Stránka projektu



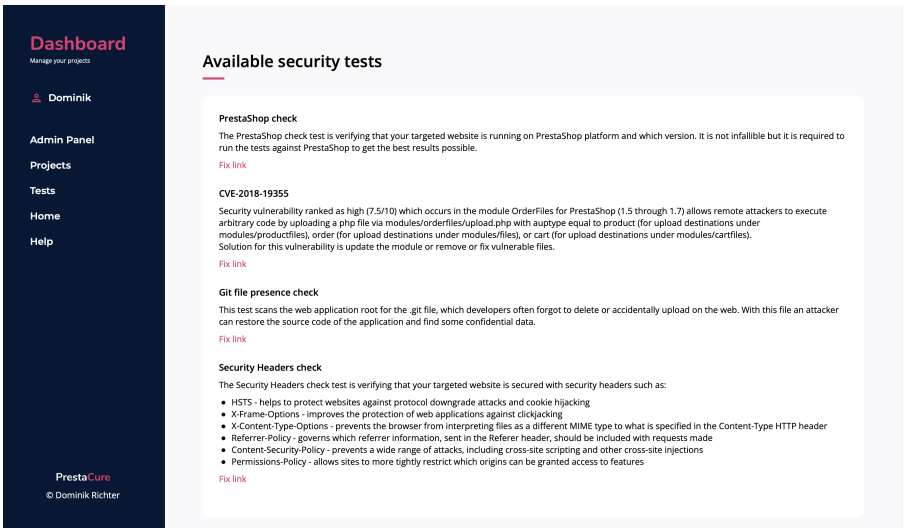
Obr. A.4: Grafický návrh stránky konkrétního projektu

A.5 Stránka administrátorského panelu



Obr. A.5: Grafický návrh stránky administrátorského panelu

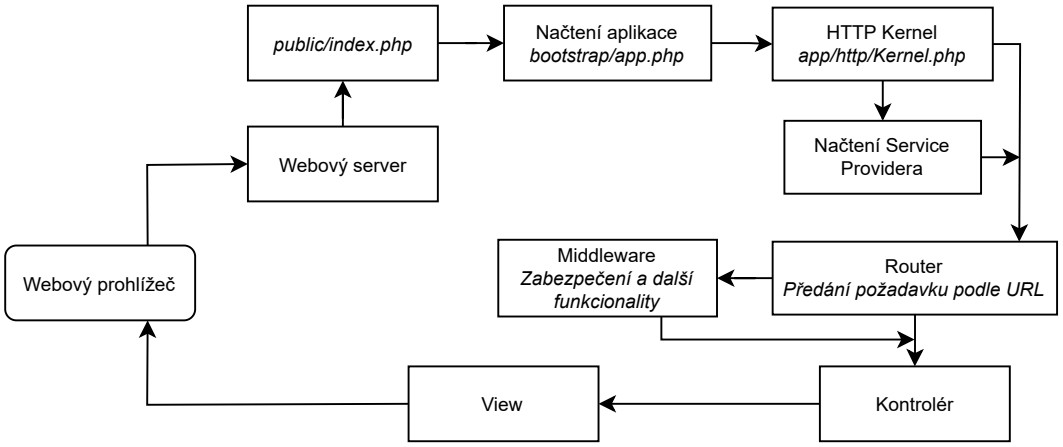
A.6 Stránka popisu penetračních testů



Obr. A.6: Grafický návrh stránky popisu penetračních testů

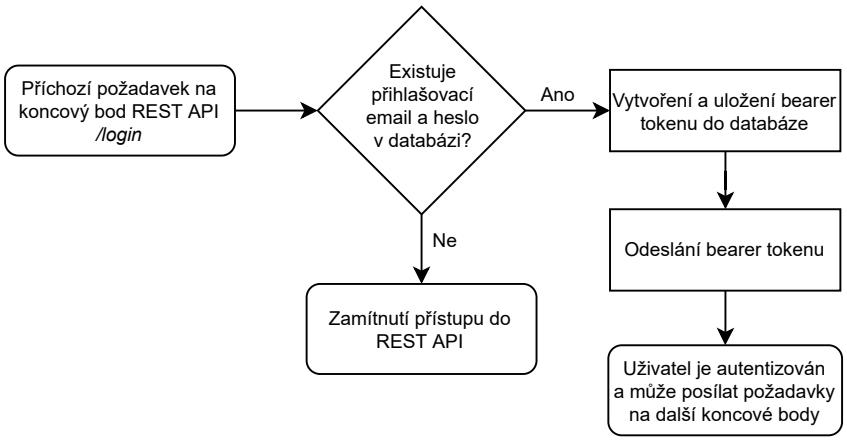
B Vývojové diagramy

B.1 Životní cyklus požadavku ve frameworku Laravel



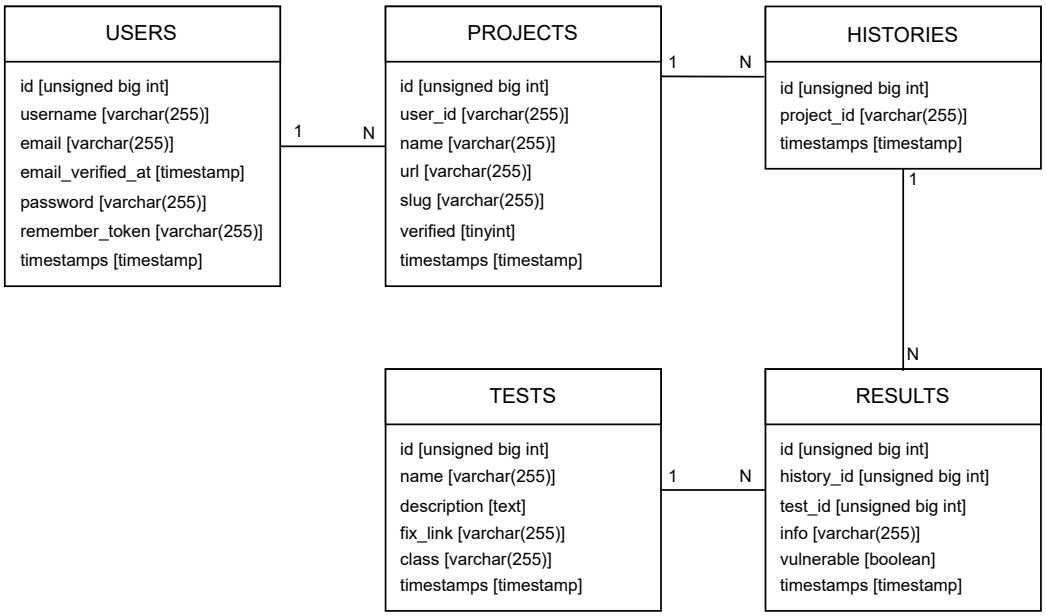
Obr. B.1: Životní cyklus požadavku ve frameworku Laravel

B.2 Přihlášení do REST API



Obr. B.2: Vývojový diagram přihlášení do REST API

C ERD diagram databáze



Obr. C.1: ERD diagram databáze

D Obsah elektronické přílohy

Elektronická příloha diplomové práce obsahuje adresář se zdrojovým kódem webové aplikace PrestaCure. Instalace a první spuštění aplikace je popsáno v kapitole 5.6. Stejný kód je také možné stáhnout z repozitáře webové služby GitHub na adrese `https://github.com/aidomon/presta-cure`. Testovací prostředí zmíněné v kapitole 4.4 je dostupné pouze na adrese `https://bit.ly/39idOmS`.