UNIVERZITA PARDUBICE

FAKULTA EKONOMICKO-SPRÁVNÍ

DIPLOMOVÁ PRÁCE

2022 Štěpán Váňa

Univerzita Pardubice
Fakulta ekonomicko-správní

Ochrana webových aplikací před kybernetickými útoky

Diplomová práce

2022 Štěpán Váňa

Univerzita Pardubice
Fakulta ekonomicko-správní
Akademický rok: 2021/2022

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Štěpán Váňa**
Osobní číslo:	**E20687**
Studijní program:	**N0688A140007 Informatika a systémové inženýrství**
Specializace:	**Informační a bezpečnostní systémy**
Téma práce:	**Ochrana webových aplikací před kybernetickými útoky**
Zadávající katedra:	**Ústav systémového inženýrství a informatiky**

Zásady pro vypracování

Cílem práce je zmapovat zranitelná místa a hrozby webových aplikací a možností ochrany před nimi. Součástí práce bude také zabezpečení vybrané webové aplikace proti kybernetickým hrozbám a otestování její odolnosti vůči těmto hrozbám.

Osnova:
- Vymezení základních pojmů.
- Zmapování zranitelných míst a hrozeb webových aplikací.
- Zmapování možností ochrany webových aplikací proti kybernetickým útokům.
- Zabezpečení vybrané webové aplikace proti kybernetickým útokům a otestování její odolnosti vůči těmto útokům.
- Formulace doporučení pro ochranu a testování webových aplikací vůči kybernetickým útokům.

Rozsah pracovní zprávy: **Cca 55 stran.**
Rozsah grafických prací:
Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam doporučené literatury:

BROOKS, Ch. J., GROW, Ch., CRAIG, P., SHORT., D. Cybersecurity essentials. Indianapolis, Indiana: Sybex, John Wiley, 2018. ISBN 978-1-119-36239-5
KOLOUCH, J., BAŠTA, P. a kol. CyberSecurity. Praha: CZ.NIC, z. s. p. o., 2019. ISBN 978-80-88168-34-8
STUTTARD, D., PINTO, M. The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws. 2nd Edition. Indianopolis: John Wiley & Sons, Inc., 2021. ISBN 978-1118026472

Vedoucí diplomové práce: **doc. Ing. Miloslav Hub, Ph.D.**
Ústav systémového inženýrství a informatiky

Datum zadání diplomové práce: **1. září 2021**
Termín odevzdání diplomové práce: **30. dubna 2022**

L.S.

prof. Ing. Jan Stejskal, Ph.D.
děkan

RNDr. Ing. Oldřich Horák, Ph.D.
vedoucí ústavu

V Pardubicích dne 1. září 2021

Prohlašuji:

Práci s názvem Ochrana webových aplikací před kybernetickými útoky jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Beru na vědomí, že v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, a směrnicí Univerzity Pardubice č. 7/2019 Pravidla pro odevzdávání, zveřejňování a formální úpravu závěrečných prací, ve znění pozdějších dodatků, bude práce zveřejněna prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne 28. 4. 2022

Štěpán Váňa

PODĚKOVÁNÍ

Tímto bych rád poděkoval doc. Ing. Miloslavu Hubovi, Ph.D., za jeho odbornou pomoc, rady a připomínky při zpracovávání diplomové práce a stejně tak za výborné vedení a dohlížení na tvorbu práce.

ANOTACE

Diplomová práce se zabývá zmapováním zranitelných míst vznikajících webových aplikací, hrozbami při jejich vývoji a možnostmi ochrany před těmito hrozbami. Součástí práce je vývoj internetového obchodu, který byl naprogramován za pomocí softwarových technologií PHP, JavaScript, HTML, SASS a SQL. Společně s jeho zabezpečením proti kybernetickým hrozbám a otestováním odolnosti proti nim za použití těchto technologií. Hlavní částí práce je aplikace ochranných prvků za účelem ochrany webové aplikace proti kybernetickým útokům. Zde jsou představeny druhy rizik pro webové aplikace a způsoby, jak je těmto hrozbám možné předejít za použití zmíněných programovacích jazyků. Výstupem práce jsou výsledky testování po aplikaci doporučených opatření a sada technik penetračních testování pro zvýšení bezpečnosti webových aplikací.

KLÍČOVÁ SLOVA

Kybernetická bezpečnost, ochrana webových aplikací, aplikace ochranných prvků, kybernetický útok, testování odolnosti, hrozby webových aplikací

TITLE

Protection of web applications against cyber attacks

ANNOTATION

The diploma thesis deals with the mapping of vulnerabilities in emerging web applications, threats in their development and possibilities of protection against these threats. Part of the work is the development of an online store, which was programmed using software technologies PHP, JavaScript, HTML, SASS and SQL. Together with its security against cyber threats and testing for resilience to them using these technologies. The main part of the work is application of security features to protect the web application against cyber attacks. Here are the types of risks for web applications and ways to avoid them using these software technologies. The output of the work are results of testing after the application of recommended measures and a set of penetration testing techniques to increase the security of web applications.

KEYWORDS

Cyber security, web application protection, security features, cyber attack, resilience testing, web application threats

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM ZKRATEK

API – Aplikační programovací prostředí (Application Programming Interface)

CSP – Zásady zabezpečení obsahu (Content Security Policy)

CSRF – Padělání požadavků napříč stránkami (Cross-Site Request Forgery)

CSS – Kaskádové styly (Cascading Style Sheets)

CVE – Běžné chyby zabezpečení a ohrožení (Common Vulnerabilities and Exposures)

DDoS – Distribuované odmítnutí služby (Distributed Denial of Service)

DMBS – Systém řízení báze dat (Database Management Systems)

DOM – Objektový model dokumentu (Document Object Model)

DoS – Odmítnutí služby (Denial of Service)

HIPPA – Zákon o přenositelnosti a odpovědnosti zdravotního pojištění (Health Insurance Portability and Accountability Act)

HTML – Hypertextový značkovací jazyk (Hypertext Markup Language)

HTTP – Aplikační protokol (HyperText Transfer Protocol)

HTTPS – Zabezpečený aplikační protokol (HyperText Transfer Protocol Secure)

IAM – Správa identit (Identity and Access Management)

ID – Identifikační znak (Identity Document)

IoT – Internet věcí (Internet of Things)

IP – Internetový protokol (Internet Protocol)

IT – Informační technologie (Information Technology)

MITM – Člověk uprostřed (Man In The Middle)

NVD – Národní databáze zranitelnosti (National Vulnerability Database)

OS – Operační systém (Operating Systém)

OWASP – Projekt o bezpečnosti webových aplikací (Open Web Application Security Project)

PDO – Datové objekty jazyka PHP (PHP Data Objects)

PHP – Hypertextový preprocesor (Hypertext Preprocessor)

SQL – Standardizovaný strukturovaný dotazovací jazyk (Structured Query Language)

SSL – Vrstva bezpečných socketů (Secure Sockets Layer)

URL – Jednotný lokátor zdroje (Uniform Resource Locator)

USB – Univerzální sériová sběrnice (Universal Seriál Bus)

VPN – Virtuální privátní síť (Virtual Private Network)

WAF – Brána firewall webové aplikace (Web Application Firewall)

WWW – Celosvětová síť (World Wide Web)

XHTML – Rozšiřitelný hypertextový značkovací jazyk (Extensible Hypertext Markup Language)

XML – Rozšiřitelný značkovací jazyk (Extensible Markup Language)

XSS – Skriptování napříč weby (Cross-Site Scripting)

ÚVOD

Jak se technologie webových aplikací vyvíjí, musí následovat robustní bezpečnostní opatření. Hrozby pro zabezpečení webových aplikací jsou realitou a vyskytují se po celém světě. Standartní opatření již nestačí, naštěstí aplikace nemusí zůstat zranitelné a čekat, až útočník využije nedostatečného opatření. Existují opatření a postupy, které lze použít k ochraně při neustále rostoucích útocích.

Škody, které může útočník způsobit mohou vést ke katastrofálním následkům. Každý rok útočníci vyvíjejí vynalézavé bezpečnostní hrozby pro webové aplikace, aby ohrozili citlivá data a získali přístup k databázi svých cílů. Aby byla zajištěna adekvátní bezpečnost proti bezpečnostním hrozbám webových aplikací, měly by podniky začlenit bezpečnostní úvahy do fáze vývoje aplikací.

Diplomová práce je zaměřená na postupy a opatření pro zvýšení bezpečnosti webových aplikací. Jsou zde představeny rizika a typy útoků, kterých útočníci využívají k dosažení svých cílů. Část práce tvoří aplikace internetového obchodu, při jejíž tvorbě byly využity teoretické znalosti k aplikování patřičných opatření k nejvyšší minimalizaci rizik a její otestování proti možným hrozbám.

1 KYBERNETICKÉ ÚTOKY NA WEBOVÉ APLIKACE

V dnešním světě, díky obrovskému pokroku internetu, můžeme na internetu najít cokoli a všechno. Mimo všech různých produktů můžeme také rezervovat služby online, a dokonce provádět platby. To vše je postaveno na moderních aplikacích, ať už se jedná o webové nebo mobilní aplikace. Protože jsme na těchto aplikacích silně závislí, nevadí nám ukládat naše osobní údaje, nebo dokonce finanční údaje, jako jsou čísla kreditních karet ve webové aplikaci. Někdy to však vede k velké ztrátě dat a reputace. [30]

Během scénáře Covid-19 jsme viděli, že internet je páteří všeho, ať už jde o schůzky v kanceláři, online kurzy, virtuální schůzky lékařů a mnoho dalšího. Jsme silně závislí na webových aplikacích, službách a produktech, které s nimi přicházejí. To také zvýšilo množství bezpečnostních hrozeb, které s tím přichází. Bezpečnost našich údajů závisí na webových aplikacích, na kterých ukládáme naše informace. V poslední době došlo k nárůstu bezpečnostních útoků, kterým nemohly uniknout ani ty největší společnosti. Několik příkladů nedávných narušení je Microsoft Exchange (březen 2021), Facebook a LinkedIn (leden a březen 2021), Clubhouse (duben 2021) nebo Bose (květen 2021). Proto je ochrana webových aplikací nanejvýš důležitá. [30]

1.1 Pojem webová aplikace

Webová aplikace je aplikační program, který je uložen na serveru a je doručován přes internet prostřednictvím rozhraní webového prohlížeče. O webovou aplikaci se jedná, pokud jakákoli komponenta webu plní nějakou funkci. Můžeme tedy říct, že obsahuje částečně nebo úplně neurčený obsah a konečný obsah stránky se určí až tehdy, když návštěvník požádá o stránku z webového serveru. Takovým webovým stránkám říkáme dynamické, protože konečný obsah se liší na základě požadavků uživatele. [15]

Aplikace mohou být navrženy pro širokou škálu použití a může je používat kdokoli, ať už se jedná o organizace či jednotlivce. Vytváří se proto, aby řešili nejrůznější úkoly a problémy. Běžně používané aplikace se využívají k vyhledávání informací; shromažďování, ukládání a analyzování dat, které poskytli návštěvníci webových aplikací; aktualizování webového místa s neustále měnícím se obsahem. Jako nejrozšířenější typy aplikací můžeme označit e-shopy, blogy nebo online kalkulačky. [15]

1.2 Princip fungování webových stránek

Webové aplikace není nutné stahovat, protože jsou přístupné prostřednictvím sítě. Uživatelé mohou přistupovat k webové aplikaci prostřednictvím webového prohlížeče, jako je Google Chrome, Mozilla Firefox nebo Safari. [32]

Aby webová aplikace fungovala, potřebuje webový server, aplikační server a databázi. Webové servery spravují požadavky, které přicházejí od klienta, zatímco aplikační server dokončí požadovanou úlohu. Databázi lze použít k uložení jakýchkoli potřebných informací. [32]

Webové aplikace mají obvykle krátké vývojové cykly a lze je vytvářet s malými vývojovými týmy. Rozlišujeme zde dva pohledy, front-end a back-end. Část webu, se kterou uživatel přímo komunikuje, se nazývá front-end. Označuje se také jako „strana klienta“ aplikace. Zahrnuje vše, co uživatelé přímo zažívají: barvy a styly textu, obrázky, grafy a tabulky, tlačítka, barvy a navigační nabídku. Jazyky používané pro programování této části mohou být HTML, CSS a JavaScript. Back-end je serverová strana webu. Ukládá a uspořádává data a také zajišťuje, že vše na klientské straně webu funguje správně. Je to část webu, kterou nevidíte a nemůžete s ní pracovat. Je to část softwaru, která nepřichází do přímého kontaktu s uživateli. K dílům a vlastnostem vyvinutým designéry back-endu mají uživatelé nepřímý přístup prostřednictvím front-endové části aplikace. K programování této části slouží mapříklad jazyky PHP, Python nebo Java. [32]

1.3 Kybernetické útoky

Kybernetický útok je útok na webovou aplikaci. Jedná se o jakýkoli pokus útočníka se zlými úmysly ohrozit zabezpečení webové aplikace. Útoky se mohou zaměřovat buď na samotnou aplikaci, aby získaly přístup k citlivým datům, nebo mohou aplikaci použít jako pracovní místo k zahájení útoků proti uživatelům aplikace. [33]

S vývojem technologie se neustále objevují nové způsoby útoků. Nyní, více než kdy jindy, jsou zapotřebí komplexní bezpečnostní nástroje. S vývojem mobilních a cloudových technologií se původní ochranné opatření stávají neúčinnými a je tak zapotřebí vývoj nových metod zabezpečení pro maximální ochranu. [33]

V dnešní době se hojně využívá nově vyvinutý přístup back-office systémů, který otevřel spoustu nových obchodních příležitostí pro organizace. Organizace nyní stále více spoléhají

na aplikační programovací prostředí (API) pro podporu inovací, rychlost vývoje a nové možnosti zpeněžení. Ovšem podle vyjádření OWASP (Open Web Application Security Project) se API stále častěji stávají cílem útoků z důvodu své povahy, díky které odhalují aplikační logiku a citlivá data. Toto přesunutí důležitých komponentů na klientskou stranu aplikace značně zvyšuje riziko útoků a vytváří nové citlivé prostředí, čímž se zvyšuje riziko útoků, jako je formjacking, manipulace DOM (Document Object Model), zneužití relace nebo zneužívaní API. Na tyto typy útoků již tradiční síťové zabezpečení nestačí a musejí být vyvíjeny nové přístupy k ochraně aplikací. [33]

Bohužel mnoho organizací stále věří, že firewall webových aplikací (WAF) je vše, co je potřeba k ochraně webových aplikací před bezpečnostními hrozbami. Ve skutečnosti WAF poskytují pouze část řešení. Tradiční bezpečnostní techniky, jako jsou WAF, nejsou schopny zastavit dnešní útoky na webové aplikace, které v mnoha případech pocházejí z prohlížeče mimo oblast zabezpečení sítě. V době, kdy jsou spuštěny, je útočník dávno pryč s citlivými informacemi a vše, co lze udělat, je zaměřit úsilí na kontrolu škod. Protože útočníci pokračují v posouvání hranic při identifikaci nových vektorů útoků, organizace musí reagovat stejně, aby zajistily, že kritická aktiva a data budou řádně a plně chráněna. [33]

Odhaduje se, že 95 % webových stránek je naprogramováno na JavaScriptu a HTML5 jazycích, které lze snadno zachytit, prohlížet a kompromitovat. Webové aplikace a rozhraní API jsou tak zranitelné vůči útokům na straně klienta, zejména pokud se spoléhají pouze na tradiční nástroje zabezpečení, jako je WAF. Podle společnosti Symantec je více než 4 800 webových stránek každý měsíc napadeno formjackingem. Útoky exfiltrace dat (neoprávněný přenos dat) na straně klienta jsou jen jedním z vektorů hrozeb, kterým webové aplikace čelí, a proto je přístup vrstveného zabezpečení tak důležitý. [33]

Jak je patrné z nedávných narušení webových aplikací u British Airways a Ticketmaster, i kdyby byl WAF na svém místě a správně nakonfigurován, nebyl by schopen zabránit těmto narušením prohlížeče/klienta. Jakmile byla zneužití objevena, stovky tisíc záznamů zákazníků již byly exfiltrovány, což už bylo příliš pozdě na podniknutí jakýchkoli kroků. [33]

Zdá se, že celková frekvence a náklady na úniky dat exponenciálně rostou. Tyto náklady jsou vysoké (přibližně 8,64 milionů USD v USA v roce 2020), protože vývojáři nejsou schopni začlenit nejnovější změny a aktualizace do svého kódu, aby překonali již zjištěné zranitelnosti. Neintuitivně má 96 % webových aplikací nějaké známé vady a anomálie. Aby byla zajištěna

adekvátní bezpečnost proti bezpečnostním hrozbám webových aplikací, měly by podniky začlenit bezpečnostní úvahy do fáze vývoje aplikací. Bohužel většina vývojářů má tendenci to odkládat až nakonec. [1]

1.4 Typy kybernetických útoků

Odborníci na kybernetickou bezpečnost rozdělují útoky do tří hlavních skupin podle toho, na co jsou zaměřeny. Rozlišujeme tak kybernetické útoky proti dostupnosti sítě, integritě sítě a důvěrnosti. Tyto útoky jsou považovány za závažné akty a podle českého práva spadají mezi trestné činy proti majetku. [21]

1.4.1 Útoky na dostupnost sítě

Aby byl informační systém užitečný, musí být dostupný oprávněným uživatelům. Opatření dostupnosti chrání včasný a nepřerušovaný přístup do systému. Některé z nejzásadnějších hrozeb dostupnosti nemají škodlivý charakter a zahrnují selhání hardwaru, neplánované výpadky softwaru a problémy s šířkou pásma sítě. Škodlivé útoky zahrnují různé formy sabotáže, které mají způsobit škodu organizaci tím, že uživatelům odepře přístup k informačnímu systému. [9]

Dostupnost a odezva webových stránek je pro mnoho podniků vysokou prioritou. Narušení dostupnosti webových stránek i na krátkou dobu může vést ke ztrátě příjmů, nespokojenosti zákazníků a poškození pověsti. Denial of Service (DoS) útok je metoda, kterou útočníci často používají k narušení webové služby. Při útoku DoS útočníci zaplaví server nadbytečnými požadavky, čímž server zahltí a znehodnotí službu pro legitimní uživatele. V druhém podobném případě se může jednat o útok DDoS. DDoS útoky jsou velmi podobné, jen si narušitel vezme na pomoc celou armádu nakažených počítačů, tzv. botnet. V botnetech se nachází několik stovek tisíc počítačů a s rozšiřováním IoT jich bude dál přibývat. Obyčejný DDoS útok lze poznat snadno, když e-shop v ČR najednou dostane několik milionů požadavků z Číny, je zřejmé, že se jedná o nelegitimní požadavky. Naštěstí je obrana díky filtraci rychlá a jednoduchá. Chytrý útočník svůj botnet schová za proxy nebo VPN, takže požadavky vypadají jako každodenní lokální provoz. Takové útoky je těžší rozeznat a nějakou dobu trvá, než se manuálně aplikuje potřebná ochrana. V průběhu let poskytovatelé služeb vyvinuli sofistikovaná protiopatření pro detekci a ochranu před DoS útoky. Nicméně útočníci také stále využívají sofistikovanějších metod a takové útoky zůstávají trvalým problémem. [9]

Protiopatření týkající se ochrany dostupnosti systému jsou stejně daleko jako ohrožení dostupnosti. Systémy, které mají vysoké požadavky na nepřetržitou dobu provozuschopnosti, by měly mít značnou hardwarovou redundanci s okamžitě dostupnými záložními servery a datovým úložištěm. U velkých podnikových systémů je běžné mít redundantní systémy na samostatných fyzických místech. Ke sledování výkonu systému a síťového provozu by měly být k dispozici softwarové nástroje. Protiopatření na ochranu před útoky DoS zahrnují firewally a směrovače. [9]

1.4.2 Útoky na integritu sítě

Opatření integrity chrání informace před neoprávněnou změnou. Tato opatření poskytují záruku přesnosti a úplnosti údajů. Potřeba chránit informace zahrnuje jak data, která jsou uložena v systémech, tak data, která jsou přenášena mezi systémy, jako je e-mail. Pro zachování integrity je nejen nutné řídit přístup na úrovni systému, ale dále zajistit, aby uživatelé systému mohli měnit pouze informace, které jsou oprávněni měnit. [9]

Ochrana integrity dat přesahuje záměrné porušení. Účinná protiopatření integrity musí také chránit před neúmyslnými změnami, jako jsou chyby uživatele nebo ztráta dat, která je důsledkem selhání systému. [9]

Zatímco všichni vlastníci systémů požadují důvěru v integritu svých dat, finanční průmysl má obzvláště důraznou potřebu zajistit, aby transakce napříč jeho systémy byly zabezpečeny proti neoprávněné manipulaci. K jednomu z nejznámějších narušení integrity finančních dat v poslední době došlo v únoru 2016, kdy kybernetičtí zloději vygenerovali podvodné výběry z účtu bangladéšské centrální banky ve Federální rezervní bance v New Yorku ve výši jedné miliardy dolarů. Útočníci vytvořili propracované schéma, které zahrnovalo získání nezbytných přihlašovacích údajů k zahájení výběrů zároveň s infikováním bankovního systému malwarem, který vymazal databázové záznamy převodů a poté potlačil potvrzovací zprávy, které by upozornily bankovní úřady na podvod. Poté, co bylo schéma odhaleno, byla většina převodů buď zablokována, nebo byly prostředky získány zpět, ale zloději byli stále schopni získat více než 60 milionů dolarů. Dalším příkladem tentokrát z ČR je ransomware, který ochromil chod Benešovské nemocnice na několik týdnů. Útok začal jediným kliknutím, a to na soubor v příloze podvrženého e-mailu. Ten bez vědomí uživatele postupně zašifroval veškeré soubory na počítači a šířil se sítí dál. Byly ochromeny servery, počítače, ale také zdravotnické přístroje. Škody dosáhly desítek milionů korun. [9]

Existuje mnoho protiopatření, která lze zavést na ochranu integrity. Kontrola přístupu a přísná autentizace mohou pomoci zabránit autorizovaným uživatelům v provádění neoprávněných změn. Ověření hash a digitální podpisy mohou pomoci zajistit, že transakce jsou autentické a že soubory nebudou změněny nebo poškozeny. Pro ochranu integrity dat jsou stejně důležité administrativní kontroly, jako je souhrn povinností a školení. [9]

1.4.3 Útoky proti důvěrnosti

Opatření v oblasti důvěrnosti chrání informace před neoprávněným přístupem a zneužitím. Většina informačních systémů obsahuje informace, které mají určitý stupeň citlivosti. Mohou to být obchodní informace, které by konkurenti mohli využít ve svůj prospěch, nebo osobní informace týkající se zaměstnanců, zákazníků nebo klientů organizace. [9]

Důvěrné informace mají často hodnotu, a proto jsou systémy často vystaveny útokům, když útočníci hledají zranitelnosti, které by mohli zneužít. Únik osobních údajů je velký problém sám o sobě. A ještě o něco větší od roku 2018, kdy vstoupilo v platnost obecné nařízení o ochraně osobních údajů (GDPR). Kromě ztráty důvěry veřejnosti firmě hrozí i pokuta v těžko představitelné výši až 20 milionů eur. Odcizené osobní údaje mají samy o sobě velkou hodnotu, útočníci je mohou zkusit prodat, použít k vydírání nebo k podvodu se zneužitím identity. Často je ale sami obratem využijí při dalším útoku založeném na phishingu nebo social engineeringu. Ne všechna porušení důvěrnosti jsou úmyslná. Mezi několik typů běžných náhodných porušení patří zasílání citlivých informací e-mailem nesprávnému příjemci, publikování soukromých dat na veřejné webové servery a ponechání důvěrných informací zobrazených na počítačovém monitoru bez přítomnosti ochrany. [9]

Zdravotnictví je příkladem odvětví, kde je povinnost chránit klientské informace velmi vysoká. Pacienti očekávají a požadují, aby poskytovatelé zdravotní péče chránili jejich soukromí, proto existují přísná nařízení upravující, jak zdravotnické organizace řídí zabezpečení. Zákon o přenositelnosti a odpovědnosti zdravotního pojištění (HIPAA) řeší bezpečnost, včetně ochrany soukromí, při nakládání s osobními zdravotními informacemi pojišťovnami, poskytovateli a zpracovateli škod. Pravidla HIPAA nařizují administrativní, fyzická a technická zabezpečení a vyžadují, aby organizace prováděly analýzu rizik. [9]

Existuje mnoho protiopatření, která organizace zavádějí, aby zajistily důvěrnost. Hesla, seznamy řízení přístupu a postupy ověřování používají software pro řízení přístupu ke zdrojům. Tyto metody řízení přístupu jsou doplněny používáním šifrování k ochraně informací,

ke kterým lze přistupovat navzdory ovládacím prvkům, jako jsou například e-maily, které jsou přenášeny. Mezi další protiopatření na ochranu důvěrnosti patří administrativní řešení, jako jsou zásady a školení, stejně jako fyzické kontroly, které lidem brání v přístupu k zařízením a vybavení. [9]

1.5 Způsoby ochrany proti kybernetickým útokům

Webovou aplikaci můžeme chránit na třech úrovních, pasivně, aktivně a v reálném čase. Pasivní ochrana je zajištěna prostřednictvím řady technik v programovacím kódu, ať už za pomocí jazyků PHP, JavaScript nebo HTML5. Zde je cílem útočníkovi znemožnit možné útoky a vytvořit pro něj prostředí, kterému je těžké porozumět a aplikovat vněm útočné akce v možných skulinách webové aplikace. [33]

Mezi pasivní ochranu patří řízení přístupu. Zde je ověřováno oprávnění a práva uživatele k určitému zdroji nebo objektu. Zdrojem je rozuměno například využití kapacity paměťového média nebo přístup k počítačovým systémům. Objektem jsou například data uložená v konkrétním adresáři nebo samotné soubory. Ověřovaným objektem může být samotný uživatel, skupina uživatelů nebo počítačové systémy. Ve skupině ověřovaných objektů může mít konkrétní uživatel jiná práva než jiný uživatel ve stejné skupině. Aby bylo možné dojít k řízení uživatelských přístupů, musí být nejprve ověřena totožnost uživatele. Nejčastějším ověřením přístupu ověření je ověřování pomocí uživatelského jména a hesla. Uživatel musí projít ověřovacím procesem, kde je požádán o zadání identifikačního jména a hesla. Tyto hodnoty jsou následně porovnány s dříve definovanými hodnotami, a pokud se údaje shodují je uživatel autentizován a následně mu jsou přidělena specifikovaná práva pro přístup k určitým funkcím, souborům apod. [20]

Aktivní ochrana slouží k ochraně před exfiltrací dat prohlížeče pomocí brány firewall v aplikaci tím, že umožňuje aplikaci nebo rozhraní API připojit se pouze k legitimním serverům a automaticky reagovat protiopatřeními. Pokud je detekována analýza kódu nebo manipulace, vypnutím funkcí webové aplikace nebo celého prohlížeče můžeme potencionálnímu útoku zabránit. [33]

Účelem firewallu je zabránit nechtěné síťové komunikaci mezi dvěma entitami. Bezpečnostní politika firewallu říká, jaká komunikace bude povolena a jaká bude zakázána. Tato pravidla jsou aplikována na pakety procházející firewallem. Firewall lze rozdělit na dva druhy podle

způsobu jejich fungování, na paketové filtry a stavové paketové filtry. Paketové filtry fungují na síťové vrstvě. Tyto filtry představují velmi rychlou a levnou variantu firewallu. Příkladem mohou být starší implementace Access Control List nebo implementace firewallu v jádru OS Linux. Stavové paketové filtry fungují podobně jako paketové filtry, navíc umožňují ukládání informací o povolených spojeních. Firewall tak nemusí vždy znovu odesílat všechny pakety do rozhodovacího procesu, ale využívá uložených informací, o již povolených spojeních a rovnou je propustí. Současné stavové paketové filtry se obvykle neomezují pouze na popsané funkce, ale přidávají možnosti kontroly obsahu paketů nebo analýzu dat konkrétního aplikačního protokolu. Pomocí toho tak například dokáží rozpoznat pokus o tunelování skrz HTTP protokolu. Speciálním druhem firewallu jsou pak aplikační brány. Tyto firewally oddělují komunikaci mezi sítěmi. Aplikační brána přijme požadavek počítačového systému, zpracuje jej a předá serveru, který vrátí odpověď aplikační bráně a ta ji následně vrátí počítačovému systému. Aplikační brána rozumí protokolu, pro který byla postavena, a dokáže v tomto protokolu detekovat nejrůznější chyby i útoky. V případě HTTP serveru tak může rozpoznat pokusy o SQL injekce, XSS útoky, nebo rozpoznat pokusy o útoky hrubou silou na uživatelské účty a takovéto dotazy rovnou zahodit. [20]

Oznámení o hrozbách v reálném čase pro upozornění podniku spočívají v tom, že pokud dojde k pokusu o manipulaci s kódem, stránkou nebo analýzou, dojde k vygenerování ohlášení, které může zahájit okamžitou provozní reakci, jako je uzavření účtů útočníků nebo aktualizace ochrany webového kódu za účelem odvrácení útoku. [33]

Do oznámení hrozeb v reálném čase spadají logy a logování. Zde dochází k zaznamenávání informací o činnosti a běhu webové aplikace. Záznamy mohou být ukládány ve formě textu do souboru nebo do databázového souboru. Úroveň detailu logování je dána možnostmi dané aplikace či systému. Z pohledu bezpečnosti musí vlastní implementaci logování předcházet rozvaha, ze které by mělo být zřejmé, jaké bezpečnostní události či bezpečnostní incidenty ohrožují aplikaci. Z této analýzy je následně možné definovat, která data a informace je potřeba logovat, aby bylo možné zjistit, že k bezpečnostní události či incidentu došlo a zároveň zdroj škodlivé aktivity. [20]

1.6 Bezpečnost serveru

Servery vyžadují zvláštní zvážení zabezpečení a umístění v rámci sítě. V síti organizace může fungovat několik různých serverů, které svým uživatelům poskytují různé služby. Ve většině případů to zahrnuje poskytování alespoň jedné kritické služby více uživatelům. Proto jsou uživatelé organizace závislí na těchto serverech při výkonu své práce. [8]

Některé ze síťových serverů, jako jsou databázové servery, mohou mít za úkol ukládat velké množství dat. Některá nebo všechna tato data mohou obsahovat kritické nebo tajné informace, jako jsou důvěrné uživatelské informace, včetně lékařských, finančních nebo personálních záznamů. Síťové servery lze také použít ke zpracování a ukládání chráněných organizačních informací, jako jsou obchodní tajemství, patenty, vynálezy nebo výrobní informace. [8]

Z těchto důvodů představují pro útočníky nejzajímavější cíle síťové servery. Zatímco část práce útočníků zahrnuje překonání veškerého ochranného zařízení v síti, cílem jejich operace je buď přístup k datům uloženým na serverech, nebo jejich narušení přístupu. Servery proto vyžadují zvláštní pozornost a umístění v rámci struktury zabezpečení sítě.

- Přístup ke sdíleným zdrojům serveru by měl být omezen na ty uživatele, kteří k získání takového přístupu mají jak potřebu, tak náležité oprávnění. Musí být zavedeny kontroly, které zajistí, že neoprávnění uživatelé nezískají přístup k důvěrným materiálům.

- Síťový přístup k některým typům serverů by měl být obvykle chráněn jedním nebo více firewally, které omezují provoz na serveru.

- Podsítě nebo směrovače by se měly používat k vytváření zabezpečených síťových segmentů nebo zón pro různé typy serverů. Například dané oddělení, jako je účetní oddělení, může být chráněno v rámci své vlastní zabezpečené podsítě a vlastnit své vlastní serverové zdroje oddělení.

- Vzhledem k tomu, že servery jsou často využívány pro ověřování uživatelů, mělo by být heslo serveru jako preventivní opatření hashováno.

- Kritické zdroje serveru by měly být pravidelně kontrolovány, aby se odhalily potenciální problémy, než přerostou ve skutečné problémy. [8]

2 HROZBY WEBOVÝCH APLIKACÍ

2.1 SQL injekce

SQL injekce je jedna z nejběžnějších technik používána pro útok na web. Je používaná ke zneužívání uživatelských dat prostřednictvím vstupů na webové stránce vkládáním dotazů SQL do těchto vstupů. V zásadě lze tyto příkazy použít k manipulaci webového serveru aplikace uživateli se zlými úmysly. Útok SQL injekce se skládá z vložení dotazů SQL prostřednictvím vstupních dat od klienta do aplikace. Úspěšné použití SQL injekce může číst citlivá data z databáze, upravovat data databáze (vkládat, updatovat nebo mazat data), provádět administrativní operace v databázi, jako je vypnutí systému řízení báze dat (DBMS), obnovení obsahu daného souboru přítomného v souboru DBMS systému a v některých případech zadávání příkazů operačnímu systému. Útoky SQL injekce jsou typem injekčního útoku, při kterém jsou dotazy SQL vkládány do vstupu datové roviny, aby ovlivnily provádění předdefinovaných dotazů SQL. Útok je v podstatě proveden umístěním meta znaku do datového vstupu, aby se pak do řídicí roviny umístily dotazy SQL, které tam dříve neexistovaly. Tato hrozba závisí na skutečnosti, že SQL nedělá žádný skutečný rozdíl mezi řídicí a datovou rovinou. [27]

Útoky SQL injekce umožňují útočníkům falšovat identitu, manipulovat s existujícími daty, způsobit problémy s odmítnutím, jako je zrušení transakcí nebo změna zůstatků, umožňují úplné odhalení všech dat v systému, zničení dat nebo je jinak znepřístupní a stát se administrátory databázového serveru. Závažnost útoků SQL injekce je omezena dovednostmi a představivostí útočníka a v menší míře i hloubkovými protiopatřeními, jako jsou připojení s nízkými oprávněními k databázovému serveru a tak dále. Obecně je SQL injekce považována za hrozbu s velmi závažným dopadem. [27]

Útok představuje pro webovou aplikaci několik rizik. Vzhledem k tomu, že SQL databáze obecně obsahují citlivá data, je ztráta důvěrnosti častým problémem zranitelnosti. Dalším rizikem je útok proti autentizaci, pokud jsou ke kontrole uživatelských jmen a hesel použity špatné dotazy SQL, může být možné připojit se k systému jako jiný uživatel bez předchozí znalosti hesla. Třetím rizikem je útok proti autorizaci, pokud jsou informace o autorizaci uchovávány v databázi SQL, může být možné tyto informace změnit pomocí úspěšného zneužití chyby zabezpečení vložením SQL dotazu. Tento typ útoku může také ohrozit integritu

webové aplikace, stejně jako může umožnit číst citlivé informace, je také možné provádět změny nebo dokonce mazat tyto informace pomocí útoku SQL injekce. [27]

Existují způsoby, jak k tomuto útoku přistupovat v rámci bezpečnosti webové aplikace. Ověřením uživatele, ověření vstupu od uživatele předdefinováním délky, typu vstupu, vstupního pole a ověření uživatele. Omezením přístupových práv uživatelů a definování toho, k jak velkému množství dat může z databáze přistupovat jakýkoli cizí člověk. Uživateli by v zásadě nemělo být uděleno oprávnění k přístupu ke všem informacím v databázi. [27]

2.2 Prolomení autentizace

Prolomení autentizace odkazuje na zranitelnosti nebo slabiny webové aplikace, která umožňuje útočníkům obejít zabezpečení přihlášení a získat přístup ke všem privilegiím vlastněným napadeným uživatelem. Autentizace zajišťuje, že k informacím a oprávněním ve webové aplikaci má přístup pouze ověřený uživatel. K prolomení autentizace dojde, když útočník obchází proces a vydává se za uživatele aplikace. Tyto nedostatky lze obecně rozdělit do dvou kategorií, špatné řízení relace a špatné řízení pověření. Obojí je klasifikováno jako nefunkční ověřování, protože útočníci se mohou za uživatele vydávat buď cestou ukradené ID relace nebo odcizenými přihlašovacími údaji. [31]

Útočníci používají širokou škálu strategií, aby využili těchto slabin, od obrovských útoků na správu pověření až po vysoce cílená schémata zaměřená na získání přístupu k pověřením konkrétní osoby. Na webových stránkách sociálních médií nebo online sázkových portálech je každá interakce, kterou uživatel provede se sítí, zaznamenána a zahrnuta do webové relace, kterou lze sledovat pomocí používané webové aplikace. Webová aplikace vydá uživateli pro každou návštěvu ID relaci. Toto ID je nezbytné, aby aplikace mohla komunikovat s uživatelem a reagovat na požadavky. [23]

Přihlašovací údaje oprávněných uživatelů mohou být také ukradeny pro přístup k aplikaci. Proto je správa pověření pro kybernetickou bezpečnost nanejvýš důležitá. Webová aplikace musí zajistit, aby nebyla povolena velmi běžná nebo snadná hesla, jako je „password1“ nebo „pass123“. Pokud je použití takových hesel povoleno, přispějí ke slabé správě pověření. Pokud webová aplikace není schopna ochránit uživatele před útočníky, kteří pronikají dovnitř prostřednictvím odcizených hesel, jedná se o formu nefunkční autentizace. [31]

Útočník může využít několika způsobů, jak získat přístupové údaje nebo odcizit ID relace k přístupu do aplikace. Mezi příklady nefunkční autentizace můžeme zařadit:

- **zneužití relace** – pokud se uživatel zapomene odhlásit z veřejného počítače, jakákoli jiná osoba může v relaci pokračovat pomocí stejné ID relace, které bylo dříve vytvořeno pro původního uživatele. Pokud je stejné ID vydáno před a po autentizaci, může to vést k typu nefunkčního autentizačního útoku, známému jako útoky fixace relací,

- **adresa URL relace** – v tomto příkladu se ID relace objeví v adrese URL webu a každý jednotlivec, který k adrese URL přistupuje prostřednictvím kabelové nebo bezdrátové sítě, ji může použít k předstírání identity uživatele,

- **credential stuffing** – někdy útočníci přistupují k databázi obsahující uživatelská hesla uživatelů, která nejsou zašifrována, a mohou často určit, zda jsou hesla platná a funkční. Každá zabezpečená webová aplikace musí mít protokoly, které proti takovým pokusům chrání,

- **password spraying** – týká se použití nejběžnějších a nejslabších hesel, jako „heslo“ nebo „123456“ útočníky, kteří se snaží získat přístup k zabezpečeným účtům. V důsledku toho byly zavedeny minimální požadavky na heslo, aby se takovým útokům zabránilo,

- **phishingové útoky** – útočníci se snaží získat od uživatelů jejich přihlašovací údaje zasíláním odkazů na webovou stránku, která se podobá původní webové aplikaci, aby přiměli uživatele prozradit své přihlašovací údaje, které se po zadání na falešném webu uloží do schránky a útočníci je mohou využít pro přihlášení do originální webové aplikace. Phishingovým útokům lze však snadno zabránit náležitou pečlivostí a ověřením používané webové aplikace. [31]

Ovšem existuje mnoho přístupů, jak se těmto rizikům a útokům bránit, mezi nejčastější a nejefektivnější ochrany můžeme zařadit:

- **regulovaní délky relace** – webová aplikace musí být schopna ukončit webové relace po určité době nečinnosti, která závisí na typu požadavků uživatele. Například bezpečný bankovní portál musí uživatele po několika minutách automaticky odhlásit, aby se předešlo jakémukoli riziku ukradení ID relace,

- **zlepšení správy relace** – webová aplikace musí být schopna vygenerovat nové ID relace po každou úspěšnou autentizaci. Tato ID musí být odstraněna, jakmile relace skončí, aby se zabránilo jakémukoli zneužití,

- **zabezpečení URL adres** – webové adresy URL musí být zabezpečené a nesmí obsahovat ID relace v žádné podobě,

- **více faktorové zabezpečení** – vyžaduje další přihlašovací údaje k ověření identity uživatele. Příkladem může být jednorázové heslo zaslané e-mailem nebo zprávou uživateli po úspěšném přihlášení do systému, které umožňuje ověření, že se opravdu jedná o správného uživatele,

- **zakázaní slabých hesel** – uživatelé musí mít povinnost nastavit hesla určité délky obsahující speciální znaky, písmena a čísla. Proto hesla, která nesplňují požadovanou složitost a délku, musí být automaticky odmítnuta,

- **přísný proces obnovy pověření** – proces obnovy pověření musí být přísný a musí zahrnovat vícenásobné ověřovací kontroly, aby bylo zajištěno, že takové možnosti obnovy nezneužijí útočníci,

- **bezpečné ukládání hesel** – hesla musí být zašifrována a zahashovaná. To pomáhá zpomalit útoky hrubou silou nebo jiné pokusy o infiltraci do databází hesel,

- **používání ochrany proti hrubé síle** – aplikace by měly mít nastavený maximální limit pro pokusy o přihlášení uživatele z konkrétní IP adresy, aby se zabránilo útokům hrubou silou. Každému uživateli překračujícímu tento limit musí být znemožněno provádět jakékoli další pokusy. [31]

2.3 Prolomení přístupu

Řízení přístupu, je způsob, jakým webová aplikace uděluje přístup k obsahu a funkcím některým uživatelům a jiným ne. Tyto kontroly se provádí po ověření a určují, co mohou „oprávnění“ uživatelé dělat. Model řízení přístupu webové aplikace je úzce svázán s obsahem a funkcemi, které web poskytuje. Kromě toho mohou uživatelé spadat do řady skupin nebo rolí s různými schopnostmi nebo oprávněními. V kontextu webových aplikací závisí řízení přístupu na autentizaci a správě relací. Autentizace identifikuje uživatele a potvrdí, že je tím, za koho se vydává. Správa relací identifikuje, které následné požadavky HTTP jsou prováděny stejným

uživatelem. Řízení přístupu určuje, zda má uživatel povoleno provést akci, kterou se pokouší učinit. [7]

Vývojáři často podceňují obtížnost implementace spolehlivého mechanismu řízení přístupu. Mnoho z těchto schémat nebylo záměrně navrženo, ale jednoduše se vyvinulo spolu s webovou stránkou. V těchto případech jsou pravidla řízení přístupu vložena na různá místa do celého kódu. Jak se stránka blíží k nasazení, tak se sbírka pravidel stává nepraktickou, že je téměř nemožné ji pochopit. Mnoho z těchto chybných schémat řízení přístupu není těžké objevit a využít. Často vše, co je potřeba, je vytvořit požadavek na funkce nebo obsah, který by neměl být schválen. Jakmile je chyba objevena, důsledky chybného schématu řízení přístupu mohou být zničující. Kromě prohlížení neoprávněného obsahu může být útočník schopen změnit nebo odstranit obsah, provádět neoprávněné funkce nebo dokonce převzít správu webu. [7]

Jedním specifickým typem problému řízení přístupu jsou administrativní rozhraní, která správcům webu umožňují spravovat web přes internet. Tyto funkce se často používají k tomu, aby správci stránek mohli efektivně spravovat uživatelské účty, data a obsah na svých stránkách. V mnoha případech weby podporují různé administrativní role, které umožňují více přístupů ke správě webu. Díky své síle jsou tato rozhraní často hlavními cíli útoků jak z venku, tak zevnitř. [7]

Z pohledu uživatele lze řízení přístupu rozdělit do třech kategorií. Vertikální řízení přístupu jsou mechanismy, které omezují přístup k citlivým funkcím, které nejsou dostupné jiným typům uživatelů. S vertikálním řízením přístupu mají různé typy uživatelů přístup k různým funkcím aplikace. Administrátor může být například schopen upravit nebo odstranit účet libovolného uživatele, zatímco běžný uživatel nemá k těmto akcím přístup. S horizontálním řízením přístupu mají různí uživatelé přístup k podmnožině zdrojů stejného typu. Například bankovní aplikace umožní uživateli prohlížet transakce a provádět platby ze svých vlastních účtů, ale ne z účtů jakéhokoli jiného uživatele. Kontextově závislé řízení přístupu omezuje přístup k funkcím a prostředkům na základě stavu aplikace nebo interakce uživatele s ní. Kontextově závislé řízení přístupu brání uživateli provádět akce v nesprávném pořadí. Maloobchodní web může například uživatelům bránit v úpravě obsahu jejich nákupního košíku po provedení platby. [5]

Nejdůležitějším krokem, jak se proti těmto rizikům chránit je promyslet požadavky aplikace na řízení přístupu a zachytit je v bezpečnostní politice webové aplikace. K definování pravidel řízení přístupu je důležité použít matici řízení přístupu. Bez zdokumentování bezpečnostní

politiky neexistuje žádná definice toho, co znamená být pro daný web bezpečný. Zásady by měly dokumentovat, jaké typy uživatelů mohou přistupovat k systému a k jakým funkcím a obsahu by měl mít každý z těchto typů uživatelů povolen přístup. Mechanismus řízení přístupu by měl být důkladně testován, aby bylo zajištěno, že neexistuje způsob, jak jej obejít. Toto testování vyžaduje různé účty a rozsáhlé pokusy o přístup k neautorizovanému obsahu nebo funkcím. Nejčastější problémy s řízením přístupu zahrnují:

- **nezabezpečená ID** – většina webových stránek používá nějakou formu ID, klíče nebo indexu jako způsob odkazování na uživatele, role, obsah, objekty nebo funkce. Pokud útočník dokáže uhodnout tyto ID a zadané hodnoty nejsou ověřeny, aby bylo zajištěno, že jsou pro aktuálního uživatele autorizovány, může útočník volně uplatnit schéma řízení přístupu, aby zjistil, k čemu mají přístup. Webové aplikace by se při ochraně neměly spoléhat na utajení jakéhokoli ID,

- **vynucené procházení v minulosti kontroly přístupu** – mnoho webů vyžaduje, aby uživatelé prošli určitými kontrolami, než jim bude udělen přístup k určitým adresám URL, které jsou obvykle „hlouběji“ na webu. Tyto kontroly nesmí uživatel obejít, když stránku s bezpečnostní kontrolou jednoduše přeskočí,

- **procházení adresářů** (Path Traversal) – tento útok zahrnuje poskytování informací o relativní cestě (např. „../../target_dir/target_file“) jako součást žádosti o informace. Takové útoky se pokouší získat přístup k souborům, které za normálních okolností nejsou přímo přístupné nikomu, nebo by byly jinak odepřeny, pokud by byly přímo požadovány. Takové útoky lze odeslat v adresách URL i v jakémkoli jiném vstupu, který nakonec přistupuje k souboru (systémová volání a příkazy v terminále),

- **oprávnění k souborům** – mnoho webových a aplikačních serverů se spoléhá na seznamy řízení přístupu poskytované souborovým systémem základní platformy. I když jsou téměř všechna data uložena na back-endových serverech, vždy existují soubory uložené lokálně na webovém a aplikačním serveru, které by neměly být veřejně přístupné, zejména konfigurační soubory, výchozí soubory a skripty, které jsou nainstalovány na většině webových a aplikačních serverů. Pouze soubory, které jsou konkrétně určeny k prezentaci uživatelům webu, by měly být označeny jako čitelné pomocí mechanismu oprávnění operačního systému. Většina adresářů by neměla být

čitelná a jen velmi málo souborů, pokud vůbec nějaké, by měly být označené jako spustitelné,

- **ukládání do mezipaměti na straně klienta** – mnoho uživatelů přistupuje k webovým aplikacím ze sdílených počítačů umístěných v knihovnách, školách, na letištích a dalších veřejných přístupových bodech. Prohlížeče často ukládají do mezipaměti webové stránky, ke kterým se mohou dostat útočníci, aby získali přístup k jinak nepřístupným částem stránek. Vývojáři by měli používat více mechanismů, včetně záhlaví HTTP a metaznaček, aby měli jistotu, že stránky obsahující citlivé informace nebudou ukládány do mezipaměti prohlížečů uživatelů. [7]

2.4 Chybná konfigurace zařízení

Útočníci se často pokoušejí zneužít neopravené chyby nebo získat přístup k výchozím účtům, nepoužívaným stránkám, nechráněným souborům a adresářům, aby získali neoprávněný přístup nebo znalost systému. [3]

K nesprávné konfiguraci zabezpečení může dojít na jakékoli úrovni zásobníku aplikací, včetně síťových služeb, platforem, webového serveru, aplikačního serveru, databáze, vlastního kódu a předinstalovaných virtuálních strojů, kontejnerů nebo úložiště. Běžná zranitelnost způsobená nesprávnou konfigurací vzniká při používání výchozích hesel, otevřených databázových instancí, zastaralých protokolů a šifrování, chybových zpráv odhalujících citlivé informace, povolení výpisu adresářů, výchozích certifikátů, nesprávně nakonfigurovaných nastavení cloudu, zbytečných funkcí, jako jsou stránky, porty a služby povolené kvůli výchozí instalaci vedoucí k nucenému procházení, vkládání příkazů, vyplňování pověření nebo výpisu adresářů. Automatizované skenery jsou užitečné pro detekci nesprávné konfigurace, použití výchozích účtů nebo konfigurací a zbytečných služeb. [3]

Tyto chyby často umožňují útočníkům neoprávněný přístup k některým systémovým datům nebo funkcím. Občas takové chyby vedou k úplné kompromitaci systému. Podle studie o porušení IBM 2020 byly špatně nakonfigurované cloudy jednou z hlavních příčin narušení dat, která stála přibližně 4,41 milionu dolarů. Obchodní dopad závisí na potřebách ochrany aplikace a dat. [24]

Aplikace může být zranitelná, pokud:

- chybí odpovídající posílení zabezpečení v jakékoli části zásobníku aplikací nebo nesprávně nakonfigurovaná oprávnění ke cloudovým službám,
- jsou povoleny nebo instalovány nepotřebné funkce (např. nepotřebné porty, služby, stránky, účty nebo oprávnění),
- výchozí účty a jejich hesla jsou stále aktivní a beze změny,
- zpracování chyb odhaluje uživatelům stopy zásobníku nebo jiné příliš informativní chybové zprávy,
- u upgradovaných systémů jsou nejnovější funkce zabezpečení deaktivovány nebo nejsou bezpečně nakonfigurovány,
- nastavení zabezpečení na aplikačních serverech, aplikačních rámcích, knihovnách, databázích a podobně nejsou nastavena na bezpečné hodnoty,
- server neodesílá bezpečnostní hlavičky nebo nejsou nastaveny na zabezpečené hodnoty,
- software je zastaralý nebo zranitelný,
- bez koordinovaného a opakovatelného procesu konfigurace zabezpečení aplikací jsou systémy vystaveny vyššímu riziku. [3]

Příkladem útočného scénáře může být aplikační server, který je dodáván s ukázkovými aplikacemi, které nejsou odebrány z produkčního serveru. Tyto ukázkové aplikace mají známé bezpečnostní chyby, které útočníci používají ke kompromitaci serveru. Pokud je jednou z těchto aplikací administrátorská konzole a výchozí účty nebyly změněny, útočník se přihlásí pomocí výchozích hesel a převezme kontrolu. Druhým příkladem je, že výpis adresáře není na serveru zakázán. Útočník zjistí, že může jednoduše vypsat adresáře. Útočník najde a stáhne zkompilované třídy, které dekompiluje a zpětně analyzuje, aby zobrazil kód. Útočník pak v aplikaci může nalézt závažnou chybu řízení přístupu. Dalším útočným scénářem je konfigurace aplikačního serveru poskytující podrobné chybové zprávy, např. trasování zásobníku, které jsou vráceny uživatelům. To může odhalit citlivé informace nebo základní chyby, jako jsou verze komponent, o kterých je známo, že jsou zranitelné. [3]

K předejití potencionálních hrozeb by měly být implementovány procesy bezpečné instalace, včetně:

- opakovatelného procesu zpevňování, který umožňuje rychlé a snadné nasazení jiného prostředí, které je správně uzamčeno. Vývojová a produkční prostředí by měla být všechna nakonfigurována identicky, s různými přihlašovacími údaji používanými v každém prostředí. Tento proces by měl být automatizován, aby se minimalizovalo úsilí potřebné k nastavení nového zabezpečeného prostředí,
- minimalizované platformy bez zbytečných funkcí, komponent, dokumentace a vzorků. S odebráním nebo neinstalováním nepoužívaných funkcí a rámců,
- úkolu zkontrolovat a aktualizovat konfigurace vhodné pro všechny bezpečnostní poznámky, aktualizace a opravy jako součást procesu správy oprav. Zejména kontrola oprávnění cloudového úložiště,
- segmentované aplikační architektury, která poskytuje efektivní a bezpečné oddělení mezi komponenty nebo tenanty pomocí segmentace, kontejnerizace nebo cloudových bezpečnostních skupin,
- zasílání bezpečnostních směrnic klientům,
- automatizovaného procesu pro ověření účinnosti konfigurací a nastavení ve všech prostředích. [3]

2.5 Cross-site Scripting

Cross-site Scripting (XSS) je útok vkládání kódu na straně klienta. Cílem útočníka je spouštět škodlivé skripty ve webovém prohlížeči zahrnutím škodlivého kódu do legitimní webové stránky nebo webové aplikace. Ke skutečnému útoku dochází, když oběť navštíví webovou stránku nebo webovou aplikaci, která spouští škodlivý kód. Webová stránka nebo webová aplikace se stává prostředkem pro doručení škodlivého skriptu do prohlížeče uživatele. Zranitelnými prostředky, které se běžně používají pro útoky Cross-site Scripting jsou fóra, nástěnky a webové stránky, které umožňují komentáře. [11]

Útočník může použít XSS k odeslání škodlivého skriptu nic netušícímu uživateli. Prohlížeč koncového uživatele nemá žádný způsob, jak zjistit, že skriptu nelze důvěřovat, a skript spustí.

Protože si myslí, že skript pochází z důvěryhodného zdroje, může škodlivý skript přistupovat ke všem souborům cookie, tokenům relace nebo jiným citlivým informacím, které prohlížeč uchovává a používá s tímto webem. Tyto skripty mohou dokonce přepsat obsah stránky HTML. Nejběžněji se útoky provádí v jazyce JavaScript. [10]

Pokud útočník zneužil zranitelnosti skrze XSS útoku na webové stránce ke spuštění libovolného JavaScriptu v prohlížeči uživatele, byla ohrožena bezpečnost této zranitelné webové stránky nebo webové aplikace a jejích uživatelů. XSS není problém uživatele jako jakákoli jiná bezpečnostní chyba. Pokud to ovlivňuje uživatele, ovlivňuje to i společnost. Cross-site Scripting lze také použít k znehodnocení webové stránky namísto cílení na uživatele. Útočník může použít vložené skripty ke změně obsahu webové stránky nebo přesměrovat prohlížeč na jinou webovou stránku, například takovou, která obsahuje škodlivý kód. [11]

Zranitelnosti skrze XSS útokem jsou vnímány jako méně nebezpečné než například zranitelnosti skrze SQL injekce. Následky schopnosti spouštět JavaScript na webové stránce se na první pohled nemusí zdát hrozivé. Většina webových prohlížečů spouští JavaScript ve velmi přísně kontrolovaném prostředí. Navíc JavaScript má omezený přístup k operačnímu systému uživatele a souborům uživatele. JavaScript však může být stále nebezpečný, pokud je zneužit jako součást škodlivého obsahu.

- Škodlivý JavaScript má přístup ke všem objektům, ke kterým má přístup zbytek webové stránky. To zahrnuje přístup k souborům cookie uživatele. K ukládání tokenů relace se často používají soubory cookie. Pokud útočník může získat soubor cookie relace uživatele, může se vydávat za tohoto uživatele, provádět akce jménem uživatele a získat přístup k jeho citlivým datům.

- JavaScript umí číst DOM prohlížeče a provádět v něm libovolné úpravy. Naštěstí je toto možné pouze na stránce, kde běží JavaScript.

- JavaScript může používat objekt XMLHttpRequest k odesílání požadavků http s libovolným obsahem do libovolných cílů.

- JavaScript v prohlížečích může používat HTML5 API. Může například získat přístup ke geolokaci uživatele, webové kameře, mikrofonu, a dokonce i ke konkrétním souborům ze systému souborů uživatele. Většina těchto rozhraní API vyžaduje

přihlášení uživatele, ale útočník může toto omezení obejít pomocí sociálního inženýrství.

- Kombinace zmíněných rizik v kombinaci se sociálním inženýrstvím umožňují útočníkům provádět pokročilé útoky včetně krádeže souborů cookie, vysazování trojských koní, keylogging, phishing a krádeže identity. Zranitelnost XSS poskytuje perfektní základ pro eskalaci útoků na závažnější. Cross-site Scripting lze také použít ve spojení s jinými typy útoků, například Cross-Site Request Forgery (CSRF). [11]

Typický útok XSS má dvě fáze. Aby mohl útočník spustit škodlivý kód JavaScript v prohlížeči oběti, musí nejprve najít způsob, jak vložit škodlivý kód na webovou stránku, kterou oběť navštíví. Poté musí oběť navštívit webovou stránku se škodlivým kódem. Pokud je útok zaměřen na konkrétní oběti, může útočník použít sociální inženýrství nebo phishing k odeslání škodlivé adresy URL oběti. [11]

Aby bylo možné provést první krok, zranitelný web musí na své stránky přímo zahrnout vstup uživatele. Útočník pak může vložit škodlivý řetězec, který bude použit na webové stránce a prohlížeč oběti s ním bude nakládat jako se zdrojovým kódem. Existují také varianty útoků XSS, kdy útočník láká uživatele k návštěvě URL pomocí sociálního inženýrství, kde škodlivý kód je součástí odkazu, na který uživatel klikne. [11]

Následující skript je spouštěn na straně serveru, použitý skript slouží k zobrazení nejnovějšího komentáře na webové stránce:

```
print "<html>"
print "<h1>Nejnovější komentář</h1>"
print database.latestComment
print "</html>"
```

Obrázek 1: Výpis komentářů z databáze

Zdroj: [11]

Skript vezme nejnovější komentář z databáze a vloží jej na stránku HTML. Předpokládá, že vytištěný komentář se skládá pouze z textu a neobsahuje žádné HTML tagy ani jiný kód. Je zranitelný vůči XSS, protože útočník by mohl odeslat komentář, který obsahuje škodlivý obsah, například:

```
<script> totoJeXSSUtok ();</script>
```

Obrázek 2 - Skript XSS útoku

Zdroj: [11]

Webový server poskytuje uživatelům, kteří navštíví tuto webovou stránku, následující kód HTML:

```
<html>
<h1>Nejnovější komentář</h1>
<script>totoJeXSSUtok();</script>
</html>
```

Obrázek 3: Vykreslení XSS útoku na webovou stránku

Zdroj: [11]

Když se stránka načte v prohlížeči oběti, spustí se útočníkův škodlivý skript. Nejčastěji si to oběť neuvědomuje a nedokáže takovému útoku zabránit. [11]

K ochraně před XSS je potřeba vstup ošetřit. Kód aplikace by nikdy neměl odesílat data přijatá jako vstup přímo do prohlížeče, aniž by se zkontroloval, zda neobsahuje škodlivý kód. Mezi hlavní opatření náleží:

- **nedůvěřování žádnému uživatelskému vstupu** – veškeré uživatelské vstupy by měly být vnímány jako nedůvěryhodné. Jakýkoli uživatelský vstup, který se používá jako součást výstupu HTML, představuje riziko XSS. Zacházení se vstupy od ověřených nebo interních uživatelů by mělo být stejné, jako zacházení s veřejnými vstupy,
- **používání escapování/zakódování** – použití vhodné techniky escapování/zakódování podle toho, kde má být vstup uživatele použit: HTML escape, JavaScript escape, CSS escape, URL escape. Pro escapování je vhodné používat existující knihovny, nepsat vlastní, pokud to není nezbytně nutné,

- **dezinfikování HTML** – pokud uživatelský vstup musí obsahovat HTML, není možné jej escapovat/zakódovat, protože by došlo k porušení platných značek. V takových případech k analýze a vyčištění HTML je nejlepší přístup použít důvěryhodnou a ověřenou knihovnu,

- **nastavení příznaku HttpOnly** – k zmírnění následků možné chyby zabezpečení v rámci XSS útoku, lze nastavit pro soubory cookie příznak HttpOnly. Soubory cookie nebudou tak dostupné prostřednictvím JavaScriptu na straně klienta,

- **použití zásad zabezpečení obsahu** – k zmírnění následků možné chyby zabezpečení v rámci XSS útoku, jsou používány také zásady zabezpečení obsahu (CSP). CSP je hlavička odpovědi HTTP, která umožňuje deklarovat dynamické prostředky, které se mohou načítat v závislosti na zdroji požadavku,

- **pravidelné skenování** – chyby zabezpečení v rámci XSS útoku mohou být zavedeny vývojáři nebo prostřednictvím externích knihoven, modulů nebo softwaru. Webové aplikace by měli být pravidelně skenovány pomocí skeneru webových zranitelností, jako je Acunetix. [11]

2.6 Používání komponent s chybami zabezpečení

Vývoj webů exponenciálně roste, částečně díky revoluci open source. Vývojáři mohou vyvíjet bohaté a složité aplikace s malými náklady a úsilím díky integraci open source vývojových nástrojů, rámců a jazyků. Bezpečnostní výzkumníci a testeři obvykle používají automatizované nástroje k odhalování kompromitovaných nebo zranitelných komponentů a poté svá zjištění zveřejňují v nástrojích pro sledování problémů, bezpečnostních radách nebo v Národní databázi zranitelností (NVD). Každý potencionální útočník, který tyto informace najde, je může použít ke zneužití konkrétních aplikačních ploch. [18]

Tento problém je velmi rozšířený, protože většina organizací používá několik integrací s otevřeným zdrojovým kódem, které komplikují ekosystém IT. Kromě toho jsou tyto součásti třetích stran obvykle implementovány s plnými přístupovými právy, což usnadňuje jejich zneužití. [18]

Rychlá výroba a dodací lhůty charakterizují DevOps, který také přispívá ke zranitelnosti, protože týmy raději používají již existující nebo externí komponenty, než aby znovu vynalézaly

kolo. Tlak na zajištění agility také znamená, že týmy pro zabezpečení a vývoj softwaru jen zřídka kontrolují zranitelnosti komponentů vyvinuté externími zdroji. Tento problém je běžně zaznamenán v komunitě vývojářů softwaru, kteří využívají bezplatné knihovny a rámce k získání předem nakonfigurovaných šablon, díky nimž jsou webové aplikace interaktivnější. [18]

Žádná aplikace, open source nebo uzavřená, nemůže být zcela bezpečná. Bezpečnostní výzkumníci a týmy DevSecOps odhalí většinu bezpečnostních zranitelností již při vývoji aplikace. Tito odborníci obvykle dokumentují a zveřejňují svá zjištění, aby bylo možné aplikovat ošetření. Útočníci v těchto publikacích obvykle vyhledávají komponenty, jejichž známá zranitelnost nebyla opravena, nebo organizace používající zastaralé verze těchto pluginů. [18]

Používání zastaralých knihoven je také podporováno nedostatkem existujícího systému standardizace verzí. V důsledku toho většina vývojářů a vlastníků open-source nedokáže specifikovat zranitelné verze svých produktů. Knihovny navíc používají různé systémy číslování verzí, kterým nemohou rozumět všichni členové vývojářského týmu, což způsobuje matoucí stahování kódu a nasazení. Dodavatelé také používají různé nástroje pro hlášení zranitelnosti, takže týmy nemohou vyhledávat objevené zranitelnosti shromážděné v centralizovaném organizovaném fondu. [18]

Největší riziko útoku nastává v případě, pokud:

- není známa verze všech komponentů, které jsou používány, jak na straně klienta, tak na straně serveru. To zahrnuje komponenty, které jsou přímo používané, stejně jako vnořené závislosti,
- je software zranitelný, nepodporovaný nebo zastaralý. To zahrnuje OS, webový/aplikační server, systém správy databází (DBMS), aplikace, rozhraní API a všechny komponenty a knihovny,
- pravidelně nejsou vyhledávány zranitelnosti a není přihlášen odběr bulletinů zabezpečení souvisejících s komponenty, které jsou používány,

- nejsou opravovány nebo upgradovány základní platformy a frameworky včas a na základě rizik. To se běžně stává v prostředích, kdy je patchování měsíční nebo čtvrtletní úlohou, což ponechává organizace po mnoho dnů nebo měsíců zranitelné,

- vývojáři softwaru netestují kompatibilitu aktualizovaných, upgradovaných nebo opravených knihoven. [4]

Pro prevenci rizik by měly být aplikované postupy a zásady, jak k těmto externím komponentám přistupovat:

- odstranění nepoužívaných balíčků, knihoven, nepotřebných funkcí, součástí, souborů a dokumentací,

- průběžná inventarizace verzí komponentů na straně klienta i serveru a jejich závislostí pomocí nástrojů, jako jsou DependencyCheck nebo pension.js. Nepřetržité monitorování zdroje jako CVE a NVD z hlediska zranitelnosti komponentů. Použití nástrojů pro analýzu složení softwaru k automatizaci procesu. Přihlášení se k odběru e-mailových upozornění na chyby zabezpečení související s komponenty, které používáte,

- získávání komponentů pouze z oficiálních zdrojů přes zabezpečené odkazy. Upřednostňování podepsaných balíčků, pro snížení pravděpodobnosti zahrnutí upraveného škodlivého komponentu,

- monitorování knihoven a komponentů, které nejsou udržovány nebo nevytvářejí bezpečnostní patche pro starší verze. Pokud patchování není možné, mělo by se zvážit nasazení virtuálního patche ke sledování, detekci nebo ochraně proti zjištěnému problému. [4]

2.7 Odhalení citlivých dat

Citlivé údaje jsou jakékoli informace, které mají být chráněny před neoprávněným přístupem. Citlivá data mohou zahrnovat cokoli od osobních údajů, jako jsou rodná čísla, přes bankovní informace až po přihlašovací údaje. Když se k těmto datům dostane útočník v důsledku narušení dat, uživatelé jsou vystaveni riziku zneužití citlivých dat. Úniky dat, které vedou k odhalení citlivých přihlašovacích údajů, mohou přijít s náklady v milionech korun a spolu s tím zničit

reputaci společnosti. Během 21. století používání mobilních zařízení dramaticky zvýšilo používání internetu. Vystavení citlivým údajům souvisí s tím, jak společnosti zacházejí s bezpečnostními kontrolami určitých informací. Chybějící nebo špatné šifrování je jednou z nejčastějších zranitelností, která vede k odhalení citlivých dat. Útočníci obvykle využívají odhalení citlivých dat k získání hesel, kryptografických klíčů, tokenů a dalších informací, které mohou použít ke kompromitaci systému. [26]

Mezi běžně známé nedostatky, které vedou k odhalení citlivých údajů patří nedostatečné zabezpečení SSL/HTTPS na webových aplikacích nebo chyby v zabezpečení vkládání SQL dotazů v databázi. Jak webové aplikace získávají běžné využití pro moderní podniky, je důležité chránit uživatele. SSL certifikáty se používají k šifrování dat mezi webovými aplikacemi a webovými servery. Organizace s nesprávně nakonfigurovaným zabezpečením SSL/HTTPS riskují ohrožení soukromí uživatelů a integrity dat, protože mohou být snadno zachyceny při přenosu. Bez zabezpečení vkládání SQL dotazů v databázi mohou útočníci zneužít škodlivé příkazy k načtení obsahu databáze. To jim umožňuje vytvářet dotazy SQL, které jim umožňují provádět širokou škálu akcí správy databáze. Útočníci mohou získat citlivé informace, jako jsou přihlašovací údaje uživatele nebo informace o konfiguraci aplikace, které pak použijí k dalšímu pronikání a kompromitaci systému. [19]

Kdykoli organizace postrádá bezpečnostní metody, data jsou vystavena riziku zneužití. Aby se zlepšily strategie zmírňování potenciálních útoků aplikací, musí vývojové a bezpečnostní týmy nejprve pevně pochopit způsoby, jakými jsou data náchylná k expozici. Většina kybernetických útoků se zpočátku zaměřuje na slabiny, které odhalují citlivá data, aby získali více informací a podařilo se jim pochopit fungování webové aplikace. Existuje několik hrozeb, které tyto informace odhalují, ať už jsou v klidu nebo v pohybu. [26]

Data v klidu, jsou uložena v systému, ať už je to počítač nebo síť. Tato data se pokládají za méně zranitelná bez hrozby útoků, ale zato cennější. Útočníci používají různé postupy, aby se zmocnili uložených dat, často pomocí malwaru, jako jsou trojské koně nebo počítačoví červi. Oba získávají přístup do systému obsahující data přímým stahováním z infikovaného USB disku nebo kliknutím na škodlivé odkazy zasílané e-mailem. Pokud jsou data umístěna na serveru, útočníci by se mohli dostat k informacím uloženým i v souborech mimo běžné autentizované oblasti přístupu. To zvyšuje pravděpodobnost útoku na procházení adresářů nebo cest, kdy je získán přístup do neoprávněných oblastí v rámci serveru s jinak omezeným přístupem. [26]

Data v pohybu, data jsou často v pohybu a posílají příkazy a požadavky přes sítě na jiné servery, aplikace nebo uživatele. Přenášená data jsou vysoce zranitelná, zvláště když se přesouvají přes nechráněné kanály nebo do aplikačního programovacího rozhraní (API), které umožňuje aplikacím vzájemně komunikovat. Jedním z útoků, který se zaměřuje na přenos dat, je útok typu man-in-the-middle (MITM), který zachycuje provoz a monitoruje komunikaci. Mezi těmito dvěma entitami čekají kybernetičtí zločinci, kteří jsou schopni zachytit všechna data v pohybu, včetně přihlašovacích údajů. Další typ útoku využívá zranitelnosti v protokolech pro vytváření kódu SSL (Secure Socket Layer). SSL kód se používá k šifrování dat, což ztěžuje dekódování do prostého textu, pokud je zachycen. Útok na SSL může napodobovat zabezpečený skript a oklamat uživatele, aby klikli na škodlivý kód. Chyba zabezpečení v protokolech SSL by mohla ponechat prostor pro útoky vkládání kódu, jako je cross-site scripting (XSS), které mohou spouštět nebezpečné požadavky na straně prohlížeče. [26]

V dnešní době je čím dál více webových aplikací řízených informacemi, to přimělo útočníky přesunout své zaměření z webových aplikací a serverů na citlivá data. Nejlepší postupy, jak zabránit odhalení citlivých údajů zahrnují:

- **identifikování a klasifikování citlivých dat** – je důležité určit a klasifikovat citlivá data pomocí zvláštních bezpečnostních kontrol. Tato data by pak měla být filtrována podle úrovně citlivosti a poté zabezpečena příslušnými bezpečnostními kontrolami,

- **aplikování řízení přístupu** – bezpečnostní týmy by měly zaměřit svou energii na procesy autentizace, autorizace a správy relací prostřednictvím poskytování robustního mechanismu správy identit a přístupu (IAM). Při správném řízení přístupu musí organizace zajistit, aby citlivá data mohly prohlížet a upravovat pouze pověřené osoby,

- **správné šifrování dat pomocí silných a aktualizovaných protokolů** – Citlivá data by nikdy neměla být ukládána jako prostý text. Je důležité zajistit, aby přihlašovací údaje uživatele a další osobní údaje byly chráněny pomocí moderních kryptografických algoritmů,

- **ukládání hesel pomocí silných, adaptivních a salted hashovacích funkcí** – vzhledem k pokroku v bezpečnostních kontrolách útočníci také vymysleli chytré způsoby, jak získat hesla. Útočník může například použít duhovou tabulku předem vypočítaných hashů pro přístup k souboru hesel, který používá unsalted hashe. Salted hashe zvyšují

zabezpečení hesel přidáním náhodných vstupů do hashovací funkce, čímž zaručují jedinečný výstup, a jsou proto doporučovány před unsalted hashemi,

- **zakázání ukládání do mezipaměti a automatického doplňování ve formulářích** – funkce ukládání do mezipaměti a automatického doplňování pomáhají zlepšit uživatelské prostředí, ale obsahují bezpečnostní rizika, která mohou přitahovat útočníky. Útočníci se mohou spolehnout na to, že se uživatel snadno přihlásí k účtu, protože přihlašovací údaje vyplňuje funkce automatického doplňování. Mezipaměť ukládá části webových stránek pro snazší načítání při následných návštěvách, což útočníkům umožňuje použít mezipaměť k mapování pohybů uživatele. Útočníci také používají data z mezipaměti k přizpůsobení malwaru. Jako osvědčený postup se doporučuje, aby ukládání do mezipaměti a automatické doplňování formulářů byly ve výchozím nastavení zakázány a aktivovány pouze podle potřeby,

- **minimalizování datové plochy** – bezpečnostní týmy by měly snížit povrchovou plochu systému pro útok na data tím, že zváží pečlivý návrh rozhraní API a zajistí, že do odpovědí serveru bude zahrnuto pouze naprosté minimum dat. Přitom je třeba zajistit, aby odpověď serveru neodhalila informace o konfiguraci systému. Náhodné testování a filtrování dat by se mělo provádět také na straně serveru, aby se snížilo riziko, že útočníci zachytí citlivá data v nefiltrovaném přenosu. [19]

2.8 Nedostatečné protokolování a monitorování

Téměř všechny hlavní bezpečnostní incidenty pocházejí ze zneužití nedostatečného protokolování, neplánovaných bezpečnostních strategií nebo nedostatečného monitorování. Podniky používající aplikace s nedostatečnými nebo žádnými logovacími funkcemi riskují, že zmírnění daného útoku bude trvat tak dlouho až způsobí značné škody celému technologickému zásobníku. [17]

Funkce protokolování a monitorování poskytují správcům a bezpečnostním týmům nezpracovaná data o provozu, která pomáhají odhalovat potenciální hrozby identifikací neobvyklých vzorců. Tyto mechanismy jsou základními bezpečnostními pilíři, které tvoří základ robustně spravovaného bezpečnostního rámce. [17]

Při absenci pečlivě naplánovaných logovacích mechanismů organizace zmešká auditní stopu pro bezpečnostní analýzu, což poskytuje útočníkům dostatek času na další průnik do více složek ekosystému. [17]

Útoky založené na nedostatečném monitorování a protokolování zranitelností jsou obvykle hodnoceny vysoko v prevalenci, střední v příležitostech a nízké v odhalitelnosti. Zajištění, že všechny události jsou protokolovány a monitorovány, je v důsledku toho často považováno za první krok v detekci narušení. [17]

Útočníci využívají mezery v protokolování a monitorování tím, že spoléhají na skutečnost, že bezpečnostním týmům bude chvíli trvat, než útok odhalí a napraví, aby se pokusili eskalovat napadení. Základní důvod, proč je neadekvátně přihlášený systém zneužíván útočníky, je obvykle založen na následujících nedostatcích, ke kterým dochází při absenci účinného rámce protokolování a monitorování:

- nezaznamenané události a transakce,
- chybí zálohy protokolů,
- obskurní protokolování chyb,
- chybějící plány na eskalaci porušení,
- špatná správa ověřování,
- neefektivní školení o protokolování a monitorování,
- nedostatek exportů pro analýzu dat protokolu,
- špatné konfigurace softwaru. [17]

Bez protokolování kritických bezpečnostních informací nejsou bezpečnostní administrátoři upozorňováni na žádné neobvyklé události, což mění každou zranitelnost v potenciální narušení a vede k riziku dalšího útoku. Jakmile útočník získá přístup k systému, snaží se co nejvíce skrýt svou přítomnost a identitu. U systémů, které postrádají komplexní správu protokolů, se útočníci dokonce pokouší vymazat protokoly událostí, které mohou vyvolat poplach. Útočníci se poté snaží zneužít oblasti webového serveru, které byly vyvinuty a nedodržují osvědčené bezpečnostní postupy. Typické aktivní útoky začínají tím, že útočník hledá v systému slabá místa zabezpečení. Poté využívají neefektivní reakce na incidenty k prohloubení kontroly nad systémem nebo k přístupu k důležitějším datům. Vzhledem k tomu, že doba odezvy u incidentů s nedostatečným protokolováním a monitorováním je dlouhá, obvykle 150–200 dní,

mají tito aktéři hrozeb dostatek času na to, aby diskrétně otestovali privilegovanější přístup. Útočníci obvykle využívají dobře známé pokročilé útočné strategie k pokrytí většího prostoru, jakmile získají počáteční přístup. Některé z nich zahrnují útoky na hesla, zachycování a upravování zpráv mezi klientem a serverem nebo zahlcení serveru k vypnutí nebo zpomalení reakce serveru na požadavky. [17]

Pro minimalizování této hrozby jsou doporučovány tyto přístupy:

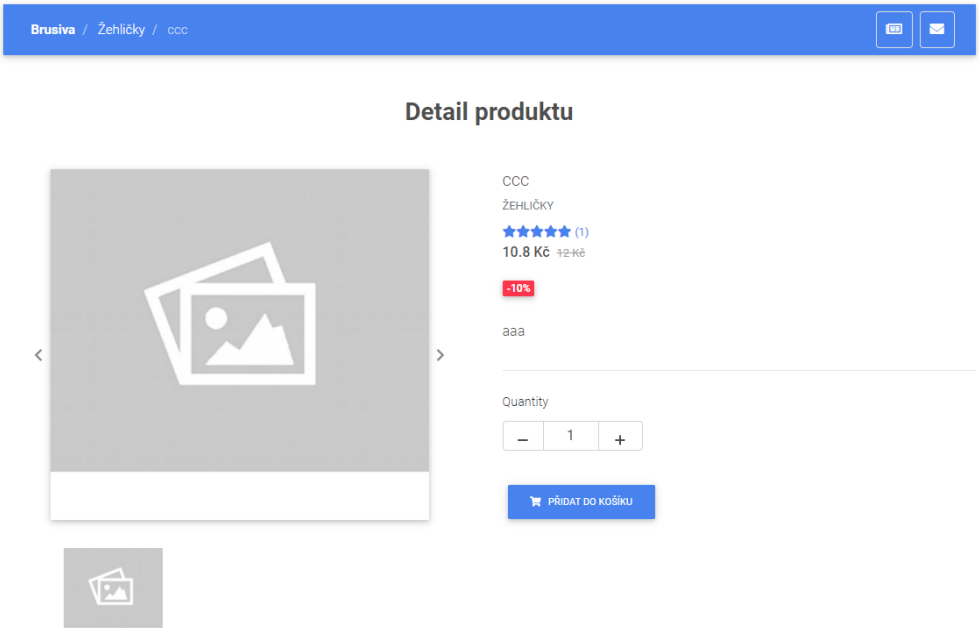
- zajištění, aby všechna selhání přihlášení, řízení přístupu a selhání ověření vstupu na straně serveru mohla být zaprotokolována s dostatečným uživatelským kontextem k identifikaci podezřelých nebo škodlivých účtů a uchována po dostatečně dlouhou dobu, aby byla umožněna opožděná analýza,
- zajištění, aby byly protokoly generovány ve formátu, který lze snadno spotřebovat centralizovanou správou protokolů,
- zajištění, aby transakce s vysokou hodnotou měly auditní záznam s kontrolami integrity, aby se zabránilo manipulaci nebo odstranění,
- zavedení efektivního monitorování a upozorňování tak, aby byly podezřelé aktivity odhaleny a reagovalo se na ně včas. [2]

3 ANALÝZA VYBRANÉ WEBOVÉ APLIKACE

V rámci diplomové práce byla naprogramována webová aplikace. Tato aplikace bude sloužit k aplikování bezpečnostních opatření a k provádění bezpečnostních testů. Vzhledem k faktu, že aplikace byla naprogramována od začátku a máme přístup k celému kódu tak aplikace bude ideálním vstupem pro aplikaci bezpečnostních prvků, ověření a porozumění bezpečnosti proti kybernetickým hrozbám.

3.1 Vybraná webová aplikace

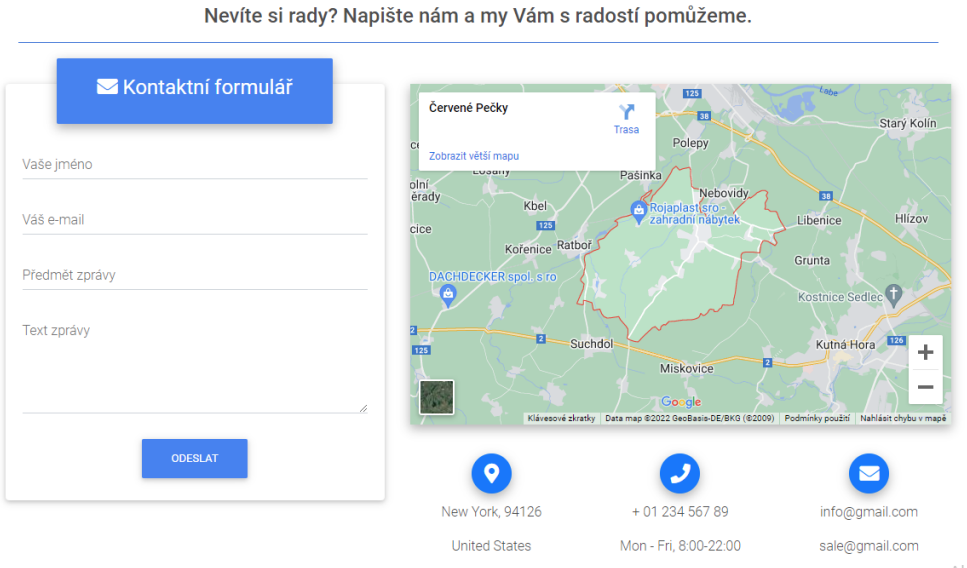
Pro záměr této práce byl naprogramován internetový obchod. Ten byl zvolen kvůli své komplexnosti, v rámci aplikace lze nalézt veškerá již zmíněná bezpečnostní rizika. Aplikace obsahuje spoustu vstupů, přes které uživatel komunikuje se serverem. To přímo přináší hrozby v podobně SQL Injection nebo XSS útoku.



Obrázek 4: Detail produktu internetového obchodu

Zdroj: vlastní zpracování

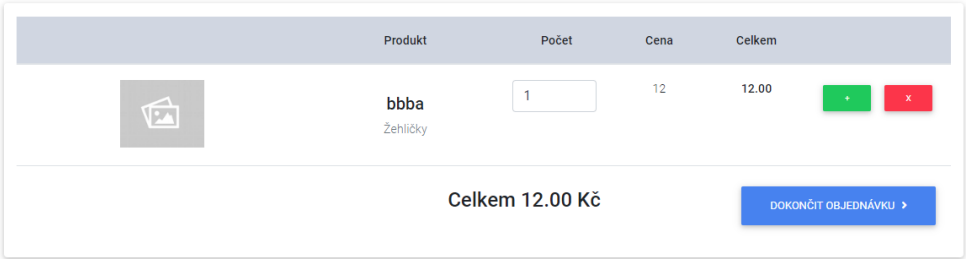
Dalšími vstupy aplikace jsou kontaktní formuláře. Zprávy skrze tyto formuláře jsou přímo ukládány do databáze a zobrazovány v administrátorském prostředí, aby kterýkoli administrátor měl možnost odpovědět, popřípadě zprávu odstranit, pokud se jedná o spam nebo jinou formu snahy zneužití aplikace.



Obrázek 5: Kontaktní formulář internetového obchodu

Zdroj: vlastní zpracování

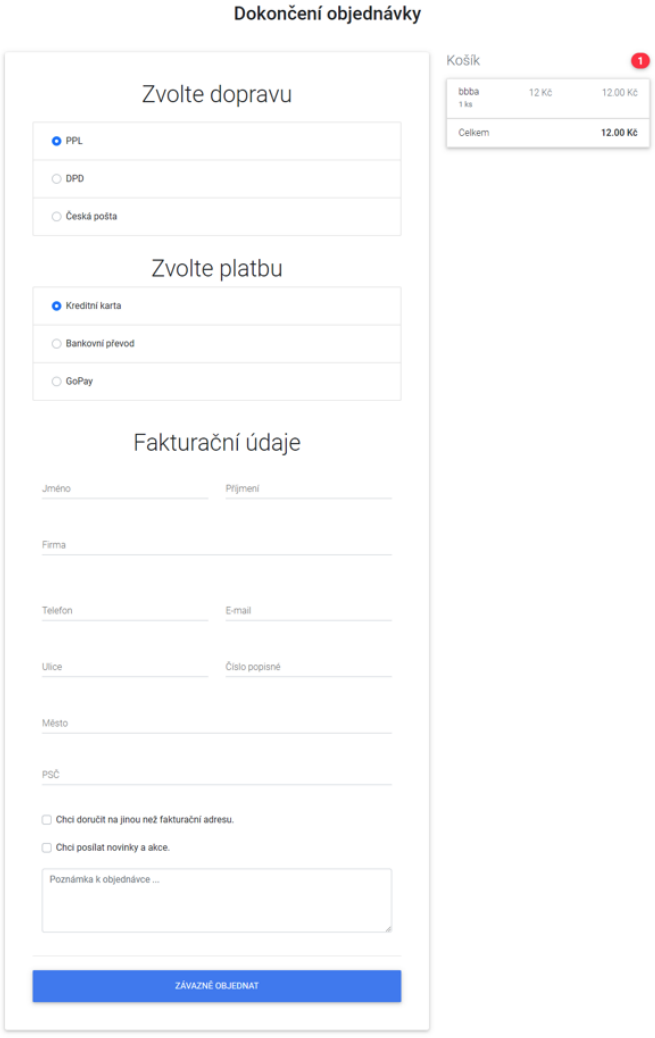
Po vybrání produktů uživatelem, které chce zakoupit jsou produkty přidány do košíku. Aplikace musí být schopná rozlišit jednotlivce a přiřadit správný nákupní košík ke správnému uživateli.



Obrázek 6: Košík internetového obchodu

Zdroj: vlastní zpracování

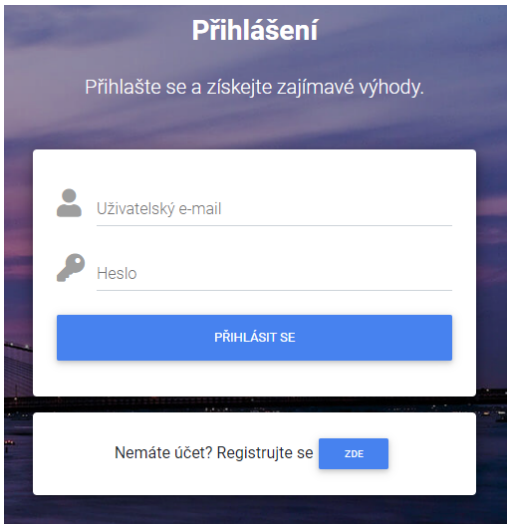
Uživatel dále postupuje k objednávkovému procesu, kde je aplikací požádán o vyplnění informací potřebných k dokončení objednávky. Toto je jeden z nejkritičtějších kroků pro správnou funkcionalitu obchodu. Pokud aplikace nebude schopná správně zaevidovat objednávku, ztrácí společnost potencionální klienty a objednávky. Data ze vstupů jsou přímo evidována do tabulek databáze, kde se nachází zbytek informací o jiných objednávkách, zákaznících a produktech. Zde je zapotřebí být velmi ostražití, aby nedošlo ke kompromitaci těchto dat v databázi. To by mělo velké následky pro společnost zejména proto, že se jedná o nejhodnotnější data.



Obrázek 7: Objednávkový proces internetového obchodu

Zdroj: vlastní zpracování

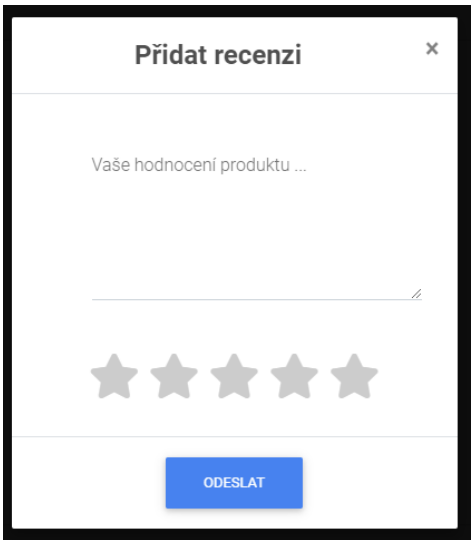
Aplikace také uživatelům nabízí možnost registrace a následné přihlášení. Tato funkcionalita přináší spoustu výhod pro společnost i zákazníky. S touto skutečností ovšem nastává spousta dalších potencionálních hrozeb. Útočníci v rámci této funkcionality mohou zaútočit a ohrozit jak společnost, tak i registrované zákazníky. Prolomení autentizace nebo přístupu jsou hlavními hrozbami, to může mít za následek odhalení citlivých dat a další poškození/zneužití systému.



Obrázek 8: Přihlašování na internetovém obchodě

Zdroj: vlastní zpracování

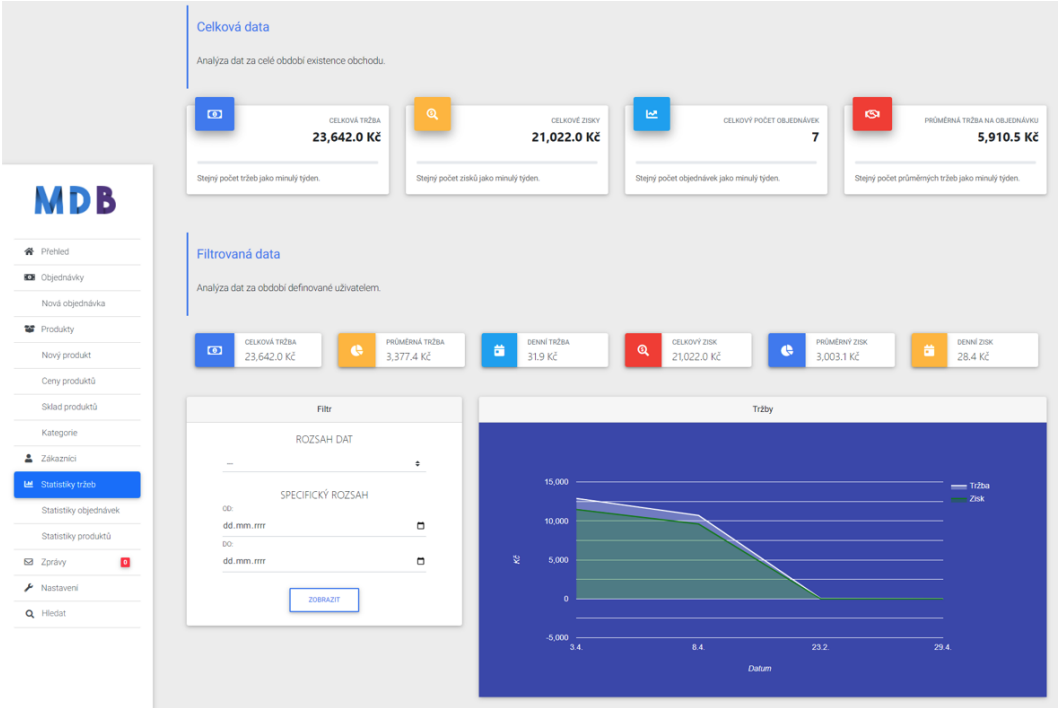
Uživatelé mají také možnost přidávání recenzí k produktům. Pro tuto funkci musí být registrování a přihlášeni. Obchod musí být schopný rozlišit přihlášeného a ověřeného uživatele, od těch, kteří tyto podmínky nesplňují.



Obrázek 9: Přidávání recenzí na internetovém obchodě

Zdroj: vlastní zpracování

Důležitou komponentou obchodu je administrátorské prostředí. V tomto prostředí mají administrátoři možnost zobrazit statistiky prodejů a veškerých finančních aktiv, evidovat objednávky, produkty, zákazníky, zprávy nebo konfigurovat základní nastavení obchodu. Kompromitace této části aplikace by měla fatální následky, proto je zapotřebí dbát zvýšené opatrnosti v aplikaci zabezpečení této části aplikace.



Obrázek 10: Administrátorské rozhraní internetového obchodu

Zdroj: vlastní zpracování

3.2 Vývoj webové aplikace

Webová aplikace internetového obchodu bylo naprogramována v lokálním prostředí, za pomocí použití následujících technologií:

- **HTML** – Hypertext Markup Language je značkovací jazyk používaný pro tvorbu webových stránek, které jsou propojeny hypertextovými odkazy. HTML je hlavním z jazyků pro vytváření stránek v systému WWW (World Wide Web), který umožňuje publikaci dokumentů na internetu. [14]

- **SASS** – je preprocesorový skriptovací jazyk, který je interpretován nebo kompilován do kaskádových stylů. Slouží pro popis způsobů zobrazení elementů na stránkách napsaných v jazycích HTML, XHTML a XML. [25]

- **PHP** – je skriptovací programovací jazyk. Je určený především pro programování dynamických internetových stránek a webových aplikací. PHP se používá na straně serveru a slouží tedy ke generování HTML kódu stránky, jenž pak server odesílá do prohlížeče. [29]

- **MySQL** – je otevřený systém řízení báze dat uplatňující relační databázový model, vytvořený švédskou firmou MySQL AB, nyní vlastněný společností Oracle Corporation. [22]

- **JavaScript** – je multiplatformní, objektově orientovaný, událostmi řízený skriptovací jazyk, jehož autorem je Brendan Eich z někdejší společnosti Netscape. Jeho syntaxe patří do rodiny jazyků C/C++/Java, ale JavaScript je od těchto jazyků zásadně odlišný sémanticky, jde o jiný jazyk. Na rozdíl od jazyku PHP slouží JavaScript na klientské straně aplikace a funguje až při zobrazení stránky v prohlížeči. [16]

- **Apache HTTP server** – je multiplatformní softwarový webový server, podporuje funkce programovacích jazyků na straně serveru. Používá se k poskytování webových stránek požadovaných klientskými počítači. Klienti obvykle požadují a prohlížejí webové stránky pomocí aplikací webových prohlížečů jako Google Chrome, Firefox, Opera apod. [6]

Pro spuštění dynamických webových stránek se zmíněnými komponenty v lokálním prostředí je zapotřebí instalace řešení, které poskytne lokální webový server. Pro tuto skutečnost bylo využito řešení EasyPHP Devserver & Webserver, poskytuje přizpůsobení lokálního serveru se stejnými vlastnosti jako produkční server a stává se tak ideálním prostředím pro vývoj a testování webových aplikací bez rizika kompromitace ze vnějšku.

4 APLIKACE OPATŘENÍ

Zabezpečení proti kybernetickým hrozbám je velmi důležitou součástí vývoje aplikace. Ještě před samotným spuštění aplikace na produkčním serveru je třeba aplikovat nejefektivnější záplaty k maximálnímu předejití potencionálních hrozeb. I přesto, že se v databázi nemusí nacházet velké množství dat o společnosti nebo jejich klientech může to mít zničující následky pro budoucí vývoj společnosti. Je tady zapotřebí se na tuto problematiku zaměřit již ve vývoji aplikace.

4.1 Zabezpečení vstupů proti SQL injekci

Jak již bylo zmíněno aplikace internetového obchodu obsahuje velké množství vstupů od uživatelů. Tyto vstupy mohou přímo ohrožovat bezpečnost dat v aplikaci. SQL injekce je přímou hrozbou pro narušení bezpečnosti. Je důležité se na všechny hrozby dívat ze dvou úhlů pohledu, z pohledu vývojáře, který chce těmto rizikům zabránit a z pohledu útočníka, který chce bezpečnost narušit. Je nezbytné aplikovat efektivní záplaty pro tyto hrozby a následně je co nejdůkladněji otestovat z druhého úhlu pohledu, tedy z pohledu útočníka.

Aplikování ochran

Při programování za snahou validace vstupu na straně klienta, je vždy zapotřebí zkontrolovat vstup na straně serveru. Vstupy ze strany klienta by se neměly nikdy klasifikovat jako důvěryhodné. Ověření pomocí HTML a JavaScriptu slouží spíše jako pomoc klientovi, aby nezadával nevyžádaná data.

Správný způsob, jak se vyhnout útokům SQL injekce, bez ohledu na to, jaká databáze je používána, je oddělit data od SQL, tak že data zůstanou daty a nebudou nikdy interpretována jako příkazy SQL. Nejspolehlivějším přístupem v rámci jazyka PHP je použití tzv. parametrizované dotazy (prepared statements). Jedná se o příkazy SQL, které jsou odesílány a analyzovány databázovým serverem odděleně od jakýchkoli parametrů. Tímto způsobem je nemožné, aby útočník vložil škodlivý SQL příkaz.

Existují dva způsoby, jak parametrizované dotazy v jazyce PHP aplikovat, PDO a MySQLi. MySQLi podporuje pouze MySQL databáze a může být použito jak v procedurálním, tak i objektově orientovaném prostředí. PDO podporuje i jiné typy databází použitím stejných funkcí a je vždy objektově orientované.

Prvním krokem je správné připojení aplikace k databázi. V rámci diplomové práce internetového obchodu byl zvolen PDO způsob.

```php
<?php

namespace customers;

use PDO;

class Dbh {

    $config = parse_ini_file('../cfg/app.ini');
    private $host = $config['host'];
    private $user = $config['username'];
    private $pwd = $config['password'];
    private $dbName = $config['db'];

    protected function connect() {
        $dsn = 'mysql:host=' . $this->host . ';dbname=' . $this->dbName . ';charset=utf8mb4';
        $pdo = new PDO($dsn, $this->user, $this->pwd);
        $pdo->setAttribute(PDO::ATTR_DEFAULT_FETCH_MODE, PDO::FETCH_ASSOC);
        return $pdo;
    }

}
```

Obrázek 11: Připojení aplikace k databázi

Zdroj: vlastní zpracování

Důležitým řádkem je řádek 18, který říká PDO, aby zakázalo emulované parametrizované dotazy a použilo skutečné parametrizované dotazy. To zajišťuje, že příkaz a hodnoty nejsou analyzovány PHP před jejich odesláním na server MySQL a to nedává útočníkovi šanci vložit škodlivý kód.

Po úspěšném připojení k databázi je zapotřebí ošetřit vstupy od klientů. K tomu jsou opět použity PDO parametrizované dotazy. V ukázce níže je část kódu, která se stará o zapsání recenze uživatele k produktu do databáze.

```php
protected function addReview($id, $email, $name, $rating, $review) {
    $sql = "INSERT INTO reviews(product_id, review_email, review_name, review_rating, review_text) VALUES (?,?,?,?,?)";
    $stmt = $this->connect()->prepare($sql);
    if ($stmt->execute([$id, $email, $name, $rating, $review])) {
        header("Location: ?success");
    } else {
        header('Location: ?error');
    }
}
```

Obrázek 12: PDO parametrizované dotazy

Zdroj: vlastní zpracování

Příkaz SQL, který je předán k přípravě je analyzován a zkompilován databázovým serverem. Zadáním parametrů (VALUES) pomocí „?“ je sděleno databázovému stroji, kde filtrovat, po zavolání spuštění je připravený příkaz zkombinován s hodnotami parametrů, který jsou

určeny. Důležité je, že hodnoty parametrů jsou kombinovány s kompilovaným příkazem, nikoli s řetězcem SQL. Všechny parametry, které jsou odeslány při použití parametrizovaného dotazu budou považovány pouze za řetězce (string). Pokud by ve výše uvedeném příkladu proměnná obsahovala ‘Toto je recenze produktu‘;DELETE FROM reviews, výsledkem by byl pouze řetězec textu “‘Toto je recenze produktu‘;DELETE FROM reviews“ a útok na vymazání všech dat v tabulce reviews by byl neúspěšný. Další výhodou použití parametrizovaných dotazů je, že pokud by stejný dotaz byl proveden mnohokrát ve stejné relaci, byl by analyzován a zkompilován pouze jednou, což poskytne zvýšení rychlosti aplikace. [13]

4.2 Autentizace uživatelů

Nejdůležitějším aspektem je správný proces přihlášení uživatele, tedy jeho totožnosti. Program musí identifikovat uživatele a přidělit mu správné pravomoci a totožnost. Pro minimalizování rizik je vhodné aplikovat hned několik opatření (viz. kapitola 2.2). Prvním ošetřením je regulace délky relace. Pokud je přihlášený uživatel po určitou dobu neaktivní je odhlášen ze systému. Při úspěšném přihlášení uživatele jsou nastaveny atributy relace, které zaznamenávají ID relace (fingerprint), start relace, id uživatele, e-mail uživatele, typ uživatele a také zda registrace uživatele byla potvrzena e-mailem.

```php
protected function authCustomer($uname, $upassword) {
    if (empty(trim($uname)) || empty(trim($upassword))) {
        return false;
    } else {
        $sql = "SELECT * FROM registred WHERE registred_email=?;";
        $stmt = $this->connect()->prepare($sql);
        if ($stmt->execute([$uname])) {
            $result = $stmt->fetch();
            if ($stmt->rowCount() > 0) {
                if (password_verify($upassword, $result['registred_password'])) {
                    session_start();
                    $_SESSION['fingerprint'] = md5($_SERVER['HTTP_USER_AGENT'] . PHRASE . $_SERVER['REMOTE_ADDR']);
                    if(!isset($_SESSION['start'])) {
                        $_SESSION['start'] = time();
                    }
                    $_SESSION['customerId'] = $result['registred_id'];
                    $_SESSION['customerUid'] = $result['registred_email'];
                    $_SESSION['customerType'] = 'customer';
                    if ($result['registred_confirm'] == 1) {
                        $_SESSION['customerConfirm'] = 1;
                    } else {
                        $_SESSION['customerConfirm'] = 0;
                    }
                    return true;
                } else {
                    return false;
                }
            } else {
                return false;
            }
        } else {
            return false;
        }
    }
}
```

Obrázek 13: Autentizace uživatele

Zdroj: vlastní zpracování

Vygenerováním atributu „fingerprint“ zajistíme pro každé přihlášení unikátní ID, které je zahashované pomocí algoritmu md5. Pro útočníka je tak nemožné uhádnout ID relace a zmocnit se jí pro svůj prospěch.

Útočníci mohou také použít metodu „Password Spraying“, je to metoda, kde útočný program obsahuje nejběžnější typy hesel. Pokud aplikace není chráněna před opakovanými pokusy o uhádnutí hesla, je aplikace vystavená další hrozbě. Aplikace internetového obchodu obsahuje skript, který ukládá aktivitu neúspěšných přihlášení. Pokud se uživateli nepodařilo přihlásit během třech pokusů v rozmezí třiceti minut, je uživatel zablokován a další pokus o přihlášení je mu umožněn třicet minut od prvního pokusu. Tímto způsobem je pro útočníka časově velmi náročné vyzkoušet spoustu předdefinovaných hesel a eventuálně uspět.

```php
$sql = "SELECT * FROM login_failed_attemps WHERE login_ip=? AND login_time > DateADD(mi, -30, Current_TimeStamp)";
$stmt = $this->connect()->prepare($sql);
if ($stmt->execute([$_SERVER['REMOTE_ADDR']])) {
    $result = $stmt->fetch();
    if ($stmt->rowCount() > 3) {
        return false;
    }
}
```

Obrázek 14: Protekce proti útočné metodě Password Spraying

Zdroj: vlastní zpracování

Po úspěšném přihlášení a zaznamenání parametrů je prováděna kontrola na jednotlivých stránkách, zda je uživatel přihlášen a disponuje tak přístupem k jednotlivým stránkám a funkcím. Je také zaznamenávána poslední aktivita uživatele, pokud byl uživatel neaktivní po dobu 20 minut, je automaticky odhlášen ze systému.

```php
7   if(isset($_SESSION['start']) && (time() - $_SESSION['start'] > 1200) && (time() - $_SESSION['last_activity'] > 1200)) {
8       session_unset();
9       session_destroy();
10  } else {
11      $_SESSION['last_activity'] = time();
12  }
```

Obrázek 15: Regulování délky relace

Zdroj: vlastní zpracování

Dalším zlepšením ochrany je zakázání registrace při použití slabých hesel. Při registraci je uživatel povinen vložit heslo o minimální délce 8 znaků, musí obsahovat alespoň jedno číslo a také jedno velké písmeno.

```
7   public function registerCustomer($email, $psw, $psw_confirm, $phone) {
8       if (empty(trim($email)) || empty(trim($psw)) || empty(trim($psw_confirm)) || empty(trim($phone))) {
9           header('Location: ?error_message=wrong_input');
10      } elseif (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
11          header('Location: ?error_message=wrong_email');
12      } elseif ($psw !== $psw_confirm) {
13          header('Location: ?error_message=wrong_psw');
14      } elseif (strlen($psw) < 8 && preg_match("/[^0,9]/", $psw) && ($psw == strtoupper($psw) || $psw == strtolower($psw))) {
15          header('Location: ?error_message=wrong_psw');
16      } else {
17          $result = $this->setCustomer($email, $psw, $phone);
18          return $result;
19      }
20  }
```

Obrázek 16: Zakázání slabých hesel

Zdroj: vlastní zpracování

S registrací také přímo souvisí ukládání hesel. Zadané heslo uživatelem musí být uchováváno v databázi v bezpečné formě. Při vkládání hesla do databáze musí být zahashováno.

```
$psw = password_hash($psw, PASSWORD_DEFAULT);
```

Obrázek 17: Bezpečné ukládání hesel

Zdroj: vlastní zpracování

Pro toto hashování je použitý algoritmus „bcrypt“, který je navržen tak, aby se časem konstanta měnila s přidáváním nových a silnějších algoritmů. Zahashování jednoduchého hesla jako „admin“ je výstupem řetězec znaků, který může vypadat následovně: „$2y$10$.vGA1O9wmRjrwAVXD98HNOgsNpDczlqm3Jq7KnEd1rVAGv3Fykk1a“.

4.3 Řízení přístupu

Prolomení přístupu přímo souvisí s prolomením autentizace, pokud při autentizaci není zajištěno správné přihlášení uživatele nebo nejsou zabezpečena ID relace (Obrázek 13: Autentizace uživatele), tak přímo vznikají rizika bezpečnosti v rámci prolomení přístupu. Aplikace internetového obchodu obsahuje několik funkcí a stránek, které jsou přístupné pouze pro autorizované uživatele. Příklad může být přidávání recenzí nebo přístup do administrátorského rozhraní. Funkce přidání recenze a rozhraní administrátorského prostředí jsou dostupné pouze pro ty uživatele, kteří vlastní účet a jsou úspěšně přihlášeni do systému.

Musí tedy proběhnout ověření, zda uživatelé skutečně disponují těmito vlastnostmi a pokud ano, teprve potom je jim poskytnut přístup do těchto rozhraní.

```
if ($_SESSION['fingerprint'] == md5($_SERVER['HTTP_USER_AGENT'] . "PHRASE" . $_SERVER['REMOTE_ADDR']) && $_SESSION['customerType'] == 'customer') {
```

Obrázek 18: Ověření přístupu

Zdroj: vlastní zpracování

Toto ověření otestuje, zda uživatel disponuje parametrem „fingerprint“, který mu byl přidělen při přihlášení (Obrázek 13: Autentizace uživatele) a zda se jedná o uživatele, který má status „customer“.

Dalším rizikem v této kategorii je špatné oprávnění k souborům. Na serveru se mohou nacházet konfigurační soubory, soubory s obrázky a podobně, ke kterým nechceme, aby měli uživatelé přístup k procházení adresářů. K tomu slouží soubor .htaccess, ve kterém jsou specifikována omezení k přístupu k těmto adresářům a souborům.

```
54  <Directory /Applications/MAMP/htdocs/Abrasives/www/images>
55      Options -Indexes
56  </Directory>
```

Obrázek 19: Nastavení oprávnění k souborům

Zdroj: vlastní zpracování

Vložením tohoto skriptu do souboru .htaccess je definován zákaz výpisu adresáře „images“.

Procházení adresářů (Path Traversal) je chyba zabezpečení webu, která umožňuje útočníkovi číst libovolné soubory na serveru, na kterém je spuštěna aplikace. To může zahrnovat kód a data aplikace, pověření pro systémy typu back-end a citlivé soubory operačního systému. V některých případech může být útočník schopen zapisovat do libovolných souborů na serveru, což mu umožní upravit data nebo chování aplikace, a nakonec převzít plnou kontrolu nad serverem. [12]

```
29  $basepath = '/var/www/';
30  $realBase = realpath($basepath);
31
32  $userpath = $basepath . $_GET['path'];
33  $realUserPath = realpath($userpath);
34
35  if ($realUserPath === false || strcmp($realUserPath, $realBase) !== 0 || strpos($realUserPath, $realBase . DIRECTORY_SEPARATOR) !== 0) {
36      header("Location: /");
37  }
```

Obrázek 20: Protekce procházení adresářů

Zdroj: vlastní zpracování

Funkce „realpath“ v zásadě převede poskytnutou cestu na skutečnou fyzickou cestu, zbaví se symbolických odkazů jako „/“, „../“ a podobně. Pokud tedy cesta uživatele nezačíná skutečnou základní cestou (/var/www/), pokouší se uživatel o Path Traversal a je přesměrován na bezpečnou stránku.

4.4 Konfigurace zařízení

Při připojování aplikace k databázi by měly být údaje o přístupu do databáze ukládány v samostatném konfiguračním souboru, nikoli zaznamenány přímo ve zdrojovém kódu. Konfigurační soubor by měl být uložen mimo veřejný kořenový adresář. Zde k němu nemůže přímo přistupovat uživatel webové aplikace, pokud neexistuje jiná chyba zabezpečení (např. chyba při procházení adresáře). Díky tomu je také bezpečné sdílet kód s ostatními vývojáři, kteří se podílejí na vývoji a chrání přihlašovací údaje pro případ, že by chybná konfigurace umožnila vytisknout prostý PHP kód.

```php
<?php

namespace customers;

use PDO;

class Dbh {

    $config = parse_ini_file('../cfg/app.ini');
    private $host = $config['host'];
    private $user = $config['username'];
    private $pwd = $config['password'];
    private $dbName = $config['db'];

    protected function connect() {
        $dsn = 'mysql:host=' . $this->host . ';dbname=' . $this->dbName . ';charset=utf8mb4';
        $pdo = new PDO($dsn, $this->user, $this->pwd);
        $pdo->setAttribute(PDO::ATTR_DEFAULT_FETCH_MODE, PDO::FETCH_ASSOC);
        return $pdo;
    }

}
```

Obrázek 21: Konfigurace připojení databáze

Zdroj: vlastní zpracování

Před spuštěním aplikace na produkční server je důležité projít kód aplikace a identifikovat funkce, které již nejsou potřebné. Při vývoji může dojít k tomu, že byly přidány funkce, které byly nahrazeny jiným přístupem nebo nejsou využívány. Tyto funkce by mohly obsahovat potencionální skuliny v bezpečnosti a vystavit tak aplikaci nebezpečí.

Stejným způsobem je potřeba zkontrolovat výchozí účty a jejich hesla obsažená v databázi. Při vývoji na lokálním serveru mohou být používány výchozí účty, například administrátorský

účet se jménem „admin“ a se stejným heslem „admin“. Je důležité ověřit, že tyto účty budou odstraněny před spuštěním aplikace na produkčním serveru. Z pohledu útočníka je velmi snadné tyto přístupové údaje uhodnout, což by vedlo k prolomení autentizace.

Zpracování chyb odhaluje uživatelům stopy zásobníku nebo jiné příliš informativní chybové zprávy. Při odesílání těchto chybových hlášek uživatelům je nutné zvážit, kolik informací můžeme uživatelům sdělit. Příliš informativní zprávy mohou odhalit nedostatek aplikace uživateli, který této skutečnosti může využít a zaměřit se na konkrétní problém aplikace a narušit tak její bezpečnost.

```
7   public function registerCustomer($email, $psw, $psw_confirm, $phone) {
8       if (empty(trim($email)) || empty(trim($psw)) || empty(trim($psw_confirm)) || empty(trim($phone))) {
9           header('Location: ?error_message=wrong_input');
10      } elseif (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
11          header('Location: ?error_message=wrong_email');
12      } elseif ($psw !== $psw_confirm) {
13          header('Location: ?error_message=wrong_psw');
14      } elseif (strlen($psw) < 8 && preg_match("/[^0,9]/", $psw) && ($psw == strtoupper($psw) || $psw == strtolower($psw))) {
15          header('Location: ?error_message=wrong_psw');
16      } else {
17          $result = $this->setCustomer($email, $psw, $phone);
18          return $result;
19      }
20  }
```

Obrázek 22: Zpracování chybových hlášek

Zdroj: vlastní zpracování

Každému neúspěšně dokončenému procesu jsou přiřazeny přesně specifikované chybové hlášky, které mají za úkol informovat uživatele. Hlášky by neměly obsahovat žádné informace přímo o systému.

4.5 Zabezpečení proti Cross-site Scripting

Jedná se o nezamýšlené spuštění vzdáleného kódu webovým klientem. Jakákoli webová aplikace se může vystavit XSS, pokud převezme vstup od uživatele a odešle jej přímo na webovou stránku. Pokud vstup obsahuje HTML nebo JavaScript, lze vzdálený kód spustit až když je tento obsah vykreslen webovým klientem. Každý parametr GET, POST nebo PUT a hodnota cookie může být cokoliv, a proto by měl být ověřen. Při výstupu kterékoli z těchto hodnot je potřeba je „escapovat“, aby nebyly vyhodnoceny neočekávaným způsobem. PHP poskytuje několik způsobů, jak zabezpečit výstup v závislosti na kontextu.

```php
<?php
$input_search = urlencode($_GET['search']);
?>
<a href="vyhledavani/<?php echo $input_search; ?>/nejnovejsi" class="<?php if($_GET['sort'] == 'nejnovejsi'){ echo ' active"';}?>">
    Nejnovější
</a>
```

Obrázek 23: Ošetření parametrů získaných z URL adres

Zdroj: vlastní zpracování

Při vyhledání produktu v internetovém obchodě jsou generovány odkazy, pomocí kterých se vyhledané produkty dají seřadit. Pokud bychom tento vstup neošetřili a byl vložen škodlivý kód, aplikace by ho spustila a mohla by se provést nevyžádaná operace. Pomocí funkce „urlencode“ bude potencionální škodlivý kód vypsán pouze jako textový řetězec.

Druhým způsobem je použití funkce „htmlspecialchars“, pokud je vstup přebírán od uživatele a dále vypisován, je důležité použití této funkce.

```php
<h4 class="text-left font-weight-bold dark-grey-text mb-2">
    <strong>Nejprodávanější produkty v kategorii <?php echo htmlspecialchars($_GET['category']); ?></strong>
</h4>
```

Obrázek 24: Dezinfikování HTML obsahu

Zdroj: vlastní zpracování

Pokud uživatel vloží skript do vstupu, tak bude vstup opět vypsán pouze jako textový řetězec a nikoli spuštěn jako skript.

4.6 Protokolování a monitorování

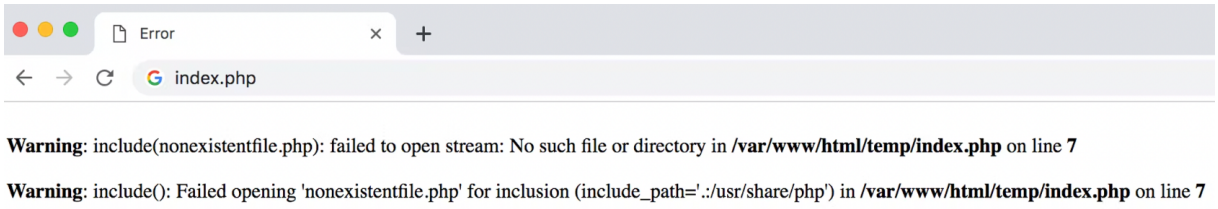
PHP obsahuje systémový nástroj pro automatické vytváření protokolů chyb. Jakmile v kódu aplikace nastane chyba, je zapsána do lokálního souboru s informacemi o ní. Je také možné vytvářet přizpůsobené protokoly voláním funkce v libovolné události aplikace. Konfigurace protokolování je obsažená v souboru „php.ini“ s následujícími parametry:

```
1   ; Log all errors
2   error_reporting = E_ALL
3   ; Don't display any errors in the browser
4   display_errors = Off
5   ; Write all logs to this file:
6   error_log = my_file.log
```

Obrázek 25: Konfigurace protokolování

Zdroj: vlastní zpracování

První parametr specifikuje zapisování veškerých chyb aplikace do souboru s protokoly. Jakmile je aplikace spuštěna na produkčním serveru, je zapotřebí vypnout chybové hlášky aplikace. Na lokálním serveru tyto hlášky pomáhají vývojářům s identifikováním chyb, ale v rámci produkčního prostředí může tato skutečnost ohrozit bezpečnost webu (Obrázek 26: Chyba zobrazená na webové aplikaci). Toto je specifikováno ve druhém parametru konfiguračního souboru. Nakonec je specifikován soubor, do kterého se protokoly budou ukládat.



Obrázek 26: Chyba zobrazená na webové aplikaci

Zdroj: vlastní zpracování

Pomocí funkce „error_log“ nyní můžeme zaznamenávat libovolnou událost. Funkce obsahuje několik parametrů, díky kterým je možné přizpůsobovat chybovou hlášku a určit kam bude odeslána.

```
39    error_log("Přístup k databázi není dostupný!", 1, "admin@admin.com");
```

Obrázek 27: Protokol funkce "error_log" odesílán e-amilem

Zdroj: vlastní zpracování

Pokud je chyba klasifikována jako chyba s vysokou prioritou, je možné protokol o chybě zaslat přímo na specifikovanou e-mailovou schránku, aby byl problém co nejrychleji napraven. V příkladu výše je funkce vyvolána, pokud dojde k chybě připojení do databáze. Prvním parametrem funkce je textový řetězec se specifikováním problému. Druhým parametrem je typ zprávy. Číslo 1 říká, že protokol o chybě bude zaslán e-mailem, do e-mailové schránky, která je specifikována jako poslední parametr. Pokud tedy dojde k chybě při připojení do databáze, bude protokol okamžitě odeslán do cílové destinace a dále mohou být zavedena opatření pro minimalizaci škod webové aplikace. Stejným způsobem lze monitorovat i jiné události, jako například přihlašování uživatelů, a zapisovat tyto události do lokálního souboru, který pak může být analyzován.

```php
$ipaddress = $_SERVER['REMOTE_ADDR'];
$webpage = $_SERVER['SCRIPT_NAME'];
$browser = $_SERVER['HTTP_USER_AGENT'];
$timestamp = date('d/m/Y h:i:s');
error_log("Neúspěšné přihlášení. IP: " . $ipaddress . ", Browser: " . $browser . ", TimeStamp: " . $timestamp . ".", 3, "/var/tmp/failed-login.log");
```

Obrázek 28: Protokol funkce "error_log" ukládán do lokálního souboru

Zdroj: vlastní zpracování

Tento protokol o chybě je vygenerován do lokálního souboru „failed-login.log“ a obsahuje informace o IP adrese uživatele, ze které byl skript spuštěn, z jakého webové prohlížeče a kdy byl skript spuštěn. Výsledný protokol o neúspěšném přihlášení vypadá následovně.

failed-login.log

```
1  Neúspěšné přihlášení. IP: 127.0.0.1, Browser: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/100.0.4896.88 Safari/537.36, TimeStamp: 18/04/2022 03:10:14.
```

Obrázek 29: Zápis protokolu v lokálním souboru

Zdroj: vlastní zpracování

5 PENETRAČNÍ TESTOVÁNÍ

5.1 Útok pomocí SQL injekce

Při testování opatření je vhodné otestovat zabezpečení několika způsoby a také veškeré potencionální vstupy pro ohrožení aplikace. V tomto případě byla otestována bezpečnost již popsaného vstupu, tedy přidávání recenzí k produktům uživateli (Obrázek 9: Přidávání recenzí na internetovém obchodě).

Zvolený útok měl za cíl vymazat celou tabulku reviews, která obsahuje recenze k produktům. Útočným vstupem byl ‘Toto je recenze produktu‘;DROP TABLE reviews, který byl vložen do textového vstupu a potvrzen tlačítkem pro potvrzení odeslání recenze.

Obrázek 30: Pokus o útok SQL injekce

Zdroj: vlastní zpracování

Ze snímku obrazovky databáze lze vidět, že dotaz pro odstranění tabulky nebyl proveden, ale byl pouze zaznamenán jako řetězec textu, který je pro aplikaci neškodný. Aplikace internetového obchodu disponuje také funkcí pro schválení recenze administrátorem. To znamená, že každá recenze musí být schválena a označena jako relevantní, aby byla zobrazena na internetovém obchodě. Tímto způsobem můžeme snadno filtrovat spamy, nevyžádané příspěvky, reklamy apod.

review_id	product_id	review_email	review_name	review_date	review_rating	review_text
7	2	2	stepanvana10@gmail.com	2022-04-20 15:57:54	5	'Toto je recenze produktu';DROP TABLE reviews

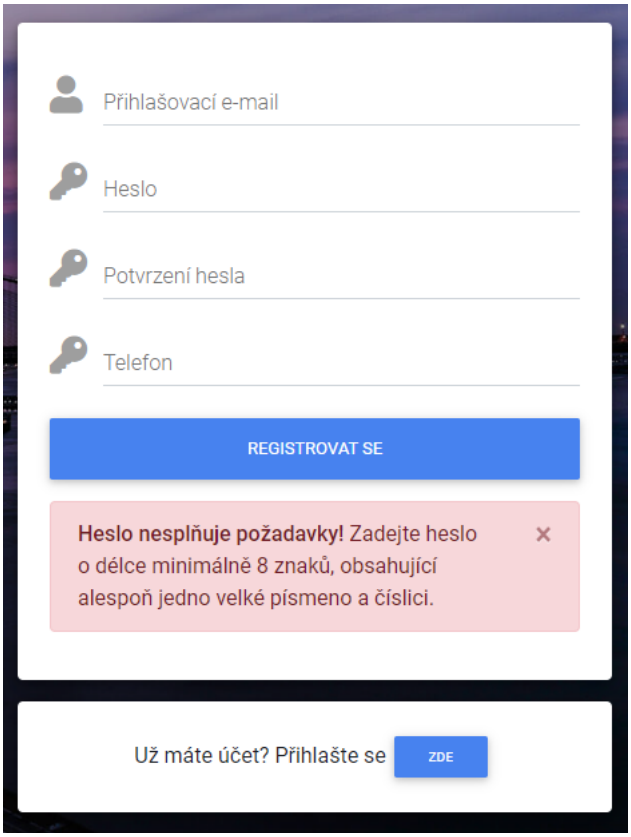
Obrázek 31: Výsledek útoku SQL injekce

Zdroj: vlastní zpracování

5.2 Útok na prolomení autentizace

5.2.1 Testování kvality hesel

Testování kvality hesel probíhá z pohledu útočníka v několika krocích. Nejdříve proběhne analýza minimálních požadavků u uživatelských hesel. Útočník se poté pokusí nastavit různé druhy slabých hesel pomocí funkcí pro vytváření hesel, aby zjistil, zda jsou tato pravidla opravdu aplikována. Po tomto testu útočník přistupuje k testování správnosti ověřování použitých hesel. Testování probíhá s vlastním vytvořeným uživatelským heslem, kde se pokouší přihlásit s různými variantami hesla, aby zjistil, zda systém ověřuje správnost přístupových údajů správným způsobem. Dochází zde k záměně velkých písmen za malé, odstranění speciálních znaků a podobně. Pokud je nějaký pokus o přihlášení s nesprávnými údaji úspěšný, pokračuje v systematickém experimentování, aby zjistil, jaké ověření se ve skutečnosti provádí. Po těchto analýzách a testech útočník identifikuje rozsah hodnot, které by útok pro uhádnutí hesla musel použít, aby byla velká pravděpodobnost úspěchu útoku. [28]

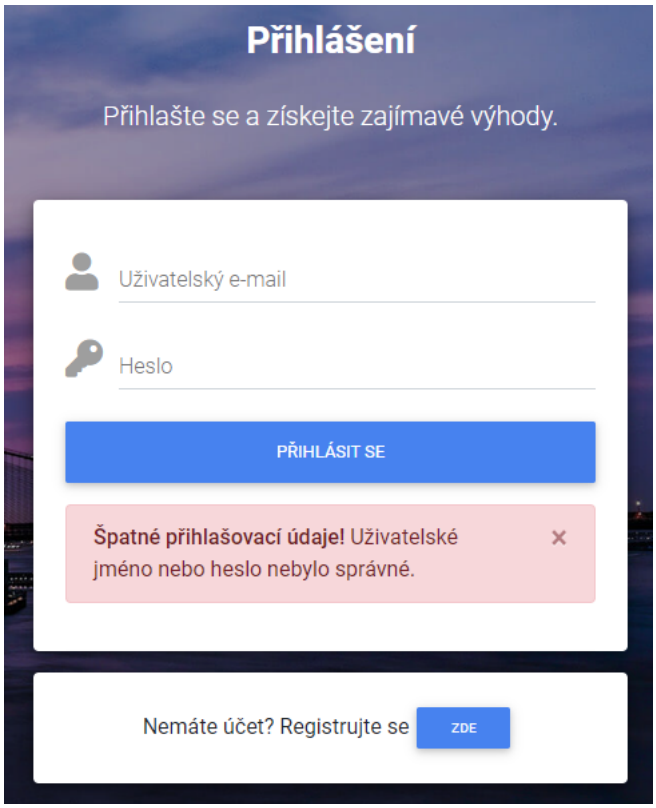


Obrázek 32: Test na kvalitu hesla

Zdroj: vlastní zpracování

5.2.2 Test na výčet uživatelských jmen

Prvním krokem je identifikování každé autentizační funkce, kde je odesíláno uživatelské jméno, včetně vstupních polí a skrytých polí formuláře. V rámci každé takové funkce jsou odeslány dva požadavky, se správným uživatelským jménem a se špatným. Následuje analýza odpovědí serveru, přesměrování, informací zobrazených na výstupu aplikace, rozdíly skryté ve zdrojovém kódu HTML a doby, kterou server potřebuje na odpověď. Například zdánlivě stejná chybová zpráva může obsahovat drobné typografické rozdíly. Jakmile jsou zjištěny nějaké rozdíly mezi odpověďmi u odeslaných správných a chybných uživatelských jmen, je test opakován s jinou dvojicí uživatelských jmen a je hledáno potvrzení, že existuje systematický rozdíl, který může poskytnout základ pro automatizovaný výčet uživatelských jmen. [28]

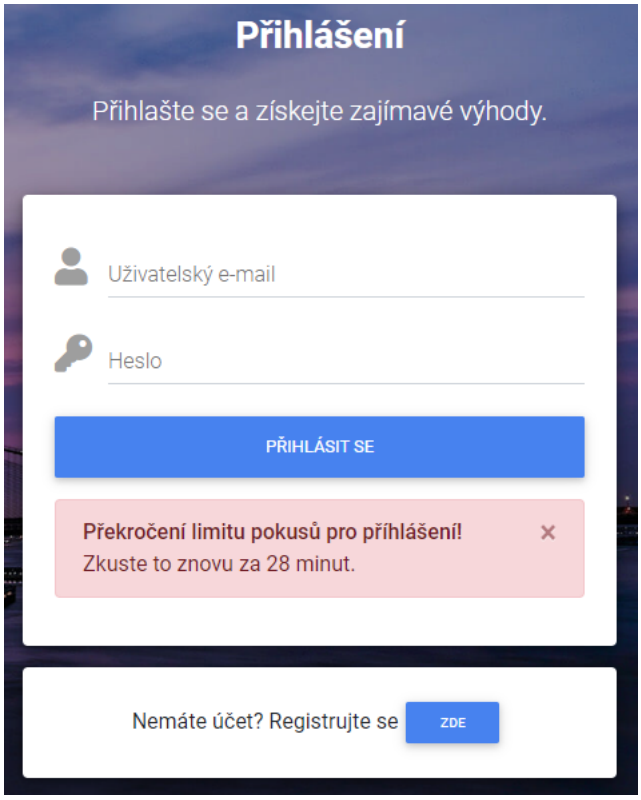


Obrázek 33: Test výčtu uživatelských jmen

Zdroj: vlastní zpracování

5.2.3 Testování odolnosti proti uhodnutí hesla

Po identifikování míst určených pro odesílání přihlašovacích údajů uživatele je na každé místo pomocí kontrolovaného účtu ručně odesíláno několik požadavků správného uživatelského jména se špatným přístupovým heslem. Během odesílání neplatných požadavků je sledována reakce aplikace a jsou identifikovány případné rozdíly. Pokud po několika neúspěšných pokusech o přihlášení nebyla zaslána chybová hláška o vyčerpání platných pokusů, je otestováno přihlášení se správnými uživatelskými údaji. V případě, že tento požadavek uspěl, je velmi pravděpodobné, že aplikace nedisponuje zásadami o uzamčení účtu. A je pak možné použít software pro uhodnutí hesla. [28]

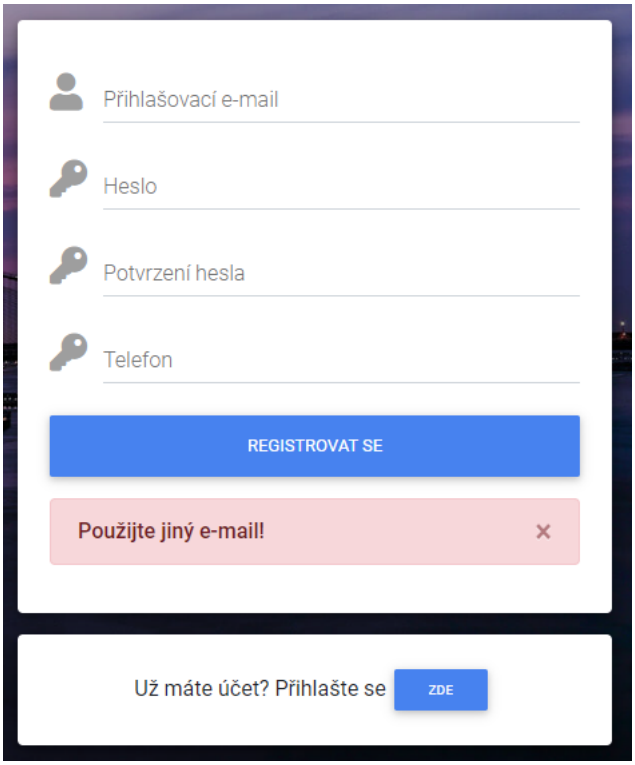


Obrázek 34: Test odolnosti proti hádání hesel

Zdroj: vlastní zpracování

5.2.4 Testování unikátních uživatelských jmen

Pokud aplikace obsahuje funkci pro vytvoření vlastního uživatelského účtu, je testována možnost vytvoření více účtů se stejnými uživatelskými jmény a jinými hesly. Jakmile aplikace zablokuje pokus o vytvoření duplicitního uživatelského jména, můžeme toto chování využít k výpisu registrovaných uživatelských jmen. Na druhou stranu, pokud aplikace zaregistruje oba účty se stejným uživatelským jménem, je testováno chování aplikace, které nastane po změně hesla pod přihlášením jednoho z uživatelských účtů. Je velmi pravděpodobné, že změna hesla pod duplicitním uživatelským jménem změní heslo pro všechny účty s tímto jménem a může dále dojít k neoprávněnému přístupu k některému z účtu. [28]



Obrázek 35: Test na unikátní uživatelská jména

Zdroj: vlastní zpracování

5.3 Útok na prolomení přístupu

5.3.1 Porozumění požadavků na řízení přístupu

Útok začíná porozuměním základní funkčnosti implementované v rámci aplikace. Ve většině aplikací jsou přítomny oba typy segregace, vertikální a horizontální. Po zmapování řízení přístupu jsou kontrolovány výsledky a zvažovány nejsnadnější cíle pro útoky na řízení přístupu. Pro nejefektivnější testování bezpečnosti je třeba získat přístup k několika účtům s různými

vertikálními a horizontálními oprávněními. Dostupnost různých druhů účtů ovlivní typy testování, které mohou být prováděny. [28]

5.3.2 Testování s více účty

V případě, kdy aplikace obsahuje vertikální segregaci oprávnění, je nejprve použit typ účtu, který má přístup k těmto funkcím. Funkce jsou analyzovány tímto účtem a následně je použit méně privilegovaný účet k pokusům získání přístupů ke každé položce funkce. Ekvivalentní test je proveden i na horizontální segregaci. Jedním účtem je testován přístup k datům druhého účtu. To obvykle zahrnuje nahrazení identifikátoru v požadavku na specifikaci zdroje patřícího jinému uživateli. Při provádění testů je potřeba otestovat každý krok vícestupňových funkcí, aby bylo ověřeno, zda byly přístupy v každé fázi správně implementovány nebo zda aplikace předpokládá, že uživatelé přistupující k pozdější fázi prošli pouze zabezpečující kontrolou. Například pokud administrativní stránka obsahující formulář kontroluje řízení přístupu i při samotném odeslání formuláře. [28]

5.3.3 Testování s omezeným přístupem

Jestliže útočník nedisponuje přístupem k účtům na různých úrovních oprávnění nebo k více účtům s přístupem k různým datům, pak testování na řízení přístupu není tak jednoduché. Mnoho běžných zranitelností bude mnohem těžší najít, protože nejsou známy názvy adres URL, identifikátory a parametry, které jsou potřebné ke zneužití skutečných slabin aplikace. [28]

K většině dat, která podléhají horizontálnímu řízení přístupu, se přistupuje pomocí identifikátoru, jako je číslo účtu nebo reference objednávky. Při testování, zda jsou kontroly přístupu účinné pouze pomocí jednoho účtu, musí být vyzkoušeno uhodnutí nebo objevení identifikátorů spojených s daty ostatních uživatelů. Pokud je to možné, bude vygenerována rychle za sebou jdoucí řada identifikátorů (například vytvořením několika nových objednávek) a učiněn pokus o identifikaci jakéhokoli vzoru, který umožní předvídat identifikátory vydané jiným uživatelům. [28]

Jakmile je nalezen způsob, jak předvídat identifikátory vydané jiným uživatelům, jsou použity techniky automatizovaného útoku ke sběru zajímavých dat patřících jiným uživatelům. [28]

5.4 Zneužití chybné konfigurace zařízení

5.4.1 Využívání chybových zpráv

Vracení podrobných chybových zpráv uživatelům může v aplikaci vyvolat nechtěné informace, které útočnici mohou využít k odhalení cenných informací. Tyto informace mohou poskytnout znalosti, které vedou k dalším útokům. Jedním z pozdějších útoků může být již zmíněný výčet uživatelských jmen. Pokud se útočník pokusí přihlásit do systému se špatným uživatelským jménem a chybová hláška zní „Toto uživatelské jméno neexistuje“, je pravděpodobné, že pokud použije správné jméno a nesprávné heslo, pak chybová hláška bude znít „Bylo zadáno špatné heslo“. Pomocí této techniky dokáže útočník vytvořit list uživatelských jmen (například e-mailů, pokud je e-mailová adresa považována pro identifikaci uživatele). [28]

5.4.2 Zranitelná konfigurace webového serveru

I ten nejjednodušší webový server přichází s množstvím konfiguračních možností, které řídí jeho chování. V minulosti bylo mnoho serverů dodáváno s nezabezpečenými výchozími možnostmi, které představují příležitosti k útoku, pokud nejsou zesíleny. [28]

Mnoho webových serverů obsahuje administrativní rozhraní, která mohou být veřejně přístupná. Ty mohou být umístěny na konkrétním místě v rámci webového kořenového adresáře nebo mohou běžet na jiném portu, jako je 8080 nebo 8443. Administrativní rozhraní mají často výchozí pověření, která jsou dobře známá a není nutné je při instalaci měnit. [28]

	USERNAME	PASSWORD
Apache Tomcat	admin	(none)
	tomcat	tomcat
	root	root
Sun JavaServer	admin	admin
Netscape Enterprise Server	admin	admin
Compaq Insight Manager	administrator	administrator
	anonymous	(none)
	user	user
	operator	operator
	user	public
Zeus	admin	(none)

Obrázek 36: Příklady výchozích přihlašovacích údajů

Zdroj: [28]

Prvním krokem útoku je zmapování aplikací a identifikování webových serverů, které mohou obsahovat přístupná administrativní rozhraní. Dále je provedeno skenování portu webového serveru a jsou identifikovány všechna administrátorská rozhraní běžící na jiném portu než hlavní cílová aplikace. Pro všechna identifikovaná rozhraní je analyzována dokumentace výrobce a seznamy běžných hesel pro získání výchozích přihlašovacích údajů. V případě, že výchozí údaje nefungují, je prováděna technika uhádnutí hesel. Pokud je získán přístup k administrátorskému rozhraní, jsou zkontrolovány funkce, zda je lze použít k dalšímu kompromitování hostitele a útoku na hlavní aplikaci. [28]

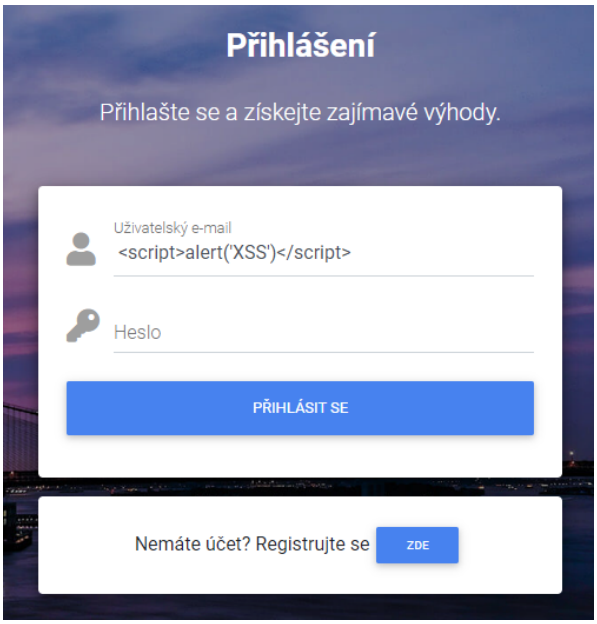
5.5 Útok pomocí Cross-site Scripting

5.5.1 Test na XSS útok

Tento útok lze provést různými způsoby. V závislosti na typu XSS útoku se škodlivý skript může projevit v prohlížeči oběti nebo uložit do databáze a spustit pokaždé, když uživatel zavolá příslušnou funkci. Hlavní příčinou tohoto útoku je nevhodná validace uživatelského vstupu, kdy se do výstupu může dostat škodlivý vstup. Útočník může zadat skript, který bude vložen do kódu webové stránky. Prohlížeč pak není schopen poznat, zda je spouštěný kód škodlivý nebo ne. V prohlížeči oběti se tedy spouští škodlivý skript nebo se uživatelům zobrazuje jakýkoli falešný formulář. Existuje několik forem, ve kterých může dojít k útokům XSS:

- U škodlivého skriptu spuštěného na straně klienta může dojít ke skriptování mezi stránkami.
- Falešná stránka nebo formulář zobrazený uživateli (kde oběť zadá přihlašovací údaje nebo klikne na škodlivý odkaz).
- Na stránkách se zobrazovanou reklamou.
- Škodlivé e-maily odeslané oběti. [28]

K tomuto útoku dochází, když útočník najde zranitelné části webu a odešle je jako vhodný škodlivý vstup. Do kódu je vkládán škodlivý skript a poté odeslán jako výstup konečnému uživateli. Pokud je pole vstupu zranitelné, jakýkoli skript bude spuštěn.

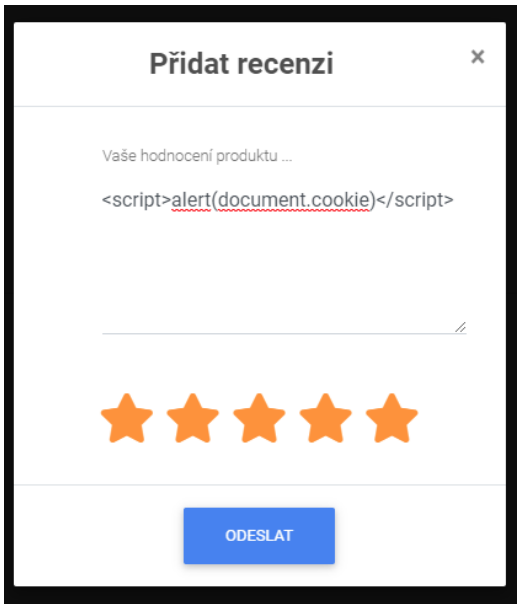


Obrázek 37: Reflected XSS útok

Zdroj: vlastní zpracování

Tento typ útok je nazýván „Reflected XSS“. K tomu dochází, když jsou škodlivé výsledky vráceny po zadání škodlivého kódu. Odražený kód XSS se neukládá trvale. V tomto případě se škodlivý kód odráží v každém výsledku webové stránky. Útočný kód lze zahrnout do falešných parametrů URL nebo HTTP. Na oběť může působit různými způsoby, zobrazením falešné škodlivé stránky nebo odesláním škodlivého e-mailu. [28]

Dalším typem XSS útoku je „Stored XSS“. Tento útok lze považovat za riskantnější a poskytuje větší poškození. Při tomto typu útoku se škodlivý kód nebo skript ukládá na webový server (například do databáze) a spouští se pokaždé, když uživatelé zavolají příslušnou funkcionalitu. Takto uložené útoky XSS mohou ovlivnit mnoho uživatelů. Vzhledem k tomu, že skript je uložen na webovém serveru, ovlivní web na delší dobu. Aby bylo možné provést uložený útok XSS, měl by být škodlivý skript odeslán prostřednictvím zranitelného vstupního formuláře (například pole komentáře nebo pole recenze). Tímto způsobem se příslušný skript uloží do databáze a provede se při načtení stránky nebo volání příslušné funkce. [28]



Obrázek 38: Stored XSS útok

Zdroj: vlastní zpracování

Skript útoku se uloží do databáze a provede se při načtení stránky, protože se na stránce zobrazí nejnovější recenze uživatele. Pokud je webová stránka zranitelná vůči XSS, zobrazí se vyskakovací okno načítání stránky s cookies. Tento skript je poměrně jednoduchý a méně škodlivý. Místo tohoto skriptu však může být zadán škodlivější kód. Například mohou být útočníkovi zaslány soubory cookies nebo se v prohlížeči oběti může zobrazit falešná stránka. [28]

Posledním typem XSS útoku je „DOM XSS“. K tomuto typu útoku dochází, když se mění prostředí DOM, ale kód na straně klienta se nemění. Při úpravě prostředí DOM v prohlížeči oběti se kód na straně klienta spustí jinak. Abychom lépe porozuměli tomu, jak se útok XSS DOM provádí, analyzujeme následující příklad. Existující webová aplikace internetového obchodu s URL https://abrasives.cz/produkt?uid=produkt1. Jak je vidět, „uid“ je parametr a „produkt1“ je jeho hodnota. Proto, abychom provedli útok XSS DOM, poslali bychom skript jako parametr. [28]

Obrázek 39: DOM XSS útok

Zdroj: vlastní zpracování

V tomto příkladu je požadavek na stránku „produkt?uid=<script>alert(document.cookie)</script>“ odeslán na abrasives.cz. Proto

pro tuto stránku prohlížeč vytváří objekt DOM, kde objekt umístění dokumentu bude obsahovat příslušný řetězec. Tímto způsobem je ovlivněno prostředí DOM. Samozřejmě místo tohoto jednoduchého skriptu může být zadáno i něco škodlivějšího. [28]

5.5.2 Path Traversal

Pokud parametr URL vypadá, že obsahuje název souboru, část názvu souboru nebo adresář je, upravena stávající hodnota parametru tak, aby byl vložen libovolný podadresář a jedna sekvence procházení. Například pokud aplikace odešle parametr „file=foo/file1.txt“ pak je změněna hodnota na „file=foo/bar/../file1.txt“. Jestliže je chování aplikace v obou případech totožné, může být zranitelná. Pokud se chování liší, může aplikace blokovat, odstraňovat nebo dezinfikovat sekvence procházení, což má za následek neplatnou cestu k souboru. [28]

V situaci, kdy je předchozí test použití předávacích sekvencí v základním adresáři úspěšný, je otestován pomocí dalších sekvencí přechod nad základní adresář a získání přístupu ke známým souborům v operačním systému serveru. Jestliže tyto pokusy selžou, aplikace může před udělením přístupu k souboru ukládat různé filtry nebo kontroly a mělo by být dále zkoumáno, jaké ovládací prvky jsou implementovány a zda existují nějaké možnosti obejití. [28]

Aplikace možná kontroluje požadovanou příponu souboru a umožňuje přístup pouze k souborům určitého druhu. Pak je otestováno použití útoku nulového bajtu nebo nového řádku spolu se známou akceptovanou příponou souboru ve snaze obejít filtr. Například „../../../../../boot.ini%00.jpg“ nebo „../../../../../etc/passwd%0a.jpg“. [28]

Aplikace možná kontroluje, zda cesta k souboru zadaná uživatelem začíná konkrétním adresářem nebo kmenem. V tomto případě je za známý přijatý kmen připojena sekvence procházení ve snaze obejít filtr. Například „/images/../../../../../../../etc/passwd“. [28]

Jestliže se podaří získat přístup pro čtení libovolných souborů na serveru, pak se útočník snaží získat některý z následujících souborů, což mu může umožnit eskalovat útok:

- Soubory hesel pro operační systém a aplikaci.
- Konfigurační soubory serveru a aplikací pro odhalení dalších zranitelností nebo vyladění jiného útoku.

- Soubory, které mohou obsahovat přihlašovací údaje k databázi.
- Zdroje dat používané aplikací, jako jsou databázové soubory MySQL nebo soubory XML.
- Zdrojový kód stránek spustitelných na serveru pro provádění kontroly kódu při hledání chyb.
- Soubory protokolu aplikací, které mohou obsahovat informace, jako jsou uživatelská jména a tokeny relace. [28]

V případě úspěchu se získáním přístupu k zápisu do libovolných souborů na serveru, je prozkoumáno, zda je možný některý z následujících útoků:

- Vytváření skriptů ve spouštěcích složkách uživatelů.
- Úprava souborů, jako je in.ftpd, aby spouštěly libovolné příkazy při příštím připojení uživatele.
- Zápis skriptů do webového adresáře s oprávněním ke spuštění a jejich volání z vašeho prohlížeče. [28]

ZÁVĚR

V diplomové práci byla popsána terminologie webových aplikací a kybernetických útoků na webové aplikace. Byly vymezeny pojmy a typy kybernetických útoků. V této části práce byly také představy základní principy ochrany proti kybernetickým útokům a bezpečnost webových serverů, na kterých aplikace operují. Pomocí rešerše odborných zdrojů byly zmapovány nejčastější hrozby webových aplikací, byly nastíněny jejich principy, příčiny a ochrany před těmito hrozbami.

V další části práce byla představena vybraná webová aplikace, která byla vyvinuta autorem od začátku za pomocí programovacích jazyků. Vybraná webová aplikace je zaměřená na internetový prodej produktů, vzhledem ke komplexnosti aplikace se jedná o ideální prostředí pro aplikování a testování kybernetických hrozeb. V rámci internetového obchodu byla vymezena funkcionalita a jeho prostředí. Také byly definovány použité technologie za účelem vývoje obchodu.

Následovala samotná aplikace ošetření webové aplikace za použití zmíněných programovacích jazyků za účelem minimalizace rizik spojených s kybernetickými útoky. Zde byly popsány konkrétní rizika aplikační struktury aplikace ve spojení s konkrétní hrozbou. Také byly vysvětleny postupy, jakými je třeba se řídit při aplikování ochran. Kde následovaly konkrétní programovací opatření, které zabraňují kybernetickým útokům za účelem poškození aplikace.

Poslední část obsahuje penetrační testování, tedy způsoby, jak se jednotlivé útoky provádí z pohledu útočníka za účelem lepšího porozumění technik a způsobů útoků. Byly popsány jednotlivé typy útoků, na jaké funkce a data se útočníci zaměřují, techniky, jak útočníci získávají data a přístup a způsoby, jak mohou narušit bezpečnost klientů aplikace.

Práce obsahuje přehled nejčastějších hrozeb kybernetických útoků a aplikace ošetření v rámci zmíněných programovacích jazyků. Vzhledem k faktu, že bezpečnost webových aplikací je velmi komplexním problémem, není prakticky možné zajistit a obsáhnout stoprocentní bezpečnost bez soustavného analyzování a implementování nových metod pro nové techniky útoků. Jak již bylo v diplomové práci zmíněno, útočníci neustále přicházejí s novými metody útoků, kterými jsou většinou o krok napřed před již známými bezpečnostními opatřeními. Jednotlivci a společnosti by tedy měli dbát na neustálé analyzování a aktualizování bezpečnostních opatření.

Námětem pro možné rozšíření práce by byla analýza organizačních, technických a fyzických opatření společně s rozšířením jiných a méně známých druhů kybernetických útoků na webové stránky či aplikace.

POUŽITÁ LITERATURA

[1] 7 Common Web Application Security Threats. *LoginRadius* [online]. 2021 [cit. 2022-04-20]. Dostupné z: https://www.loginradius.com/blog/start-with-identity/7-web-app-sec-threats

[2] A10:2017-Insufficient Logging & Monitoring. *OWASP* [online]. 2017 [cit. 2022-04-20]. Dostupné z: https://owasp.org/www-project-top-ten/2017/A10_2017-Insufficient_Logging%2526Monitoring

[3] A6:2017-Security Misconfiguration. *OWASP* [online]. 2017 [cit. 2022-04-20]. Dostupné z: https://owasp.org/www-project-top-ten/2017/A6_2017-Security_Misconfiguration

[4] A9:2017-Using Components with Known Vulnerabilities. *OWASP* [online]. 2017 [cit. 2022-04-20]. Dostupné z: https://owasp.org/www-project-top-ten/2017/A9_2017-Using_Components_with_Known_Vulnerabilities

[5] Access control vulnerabilities and privilege escalation. *PortSwigger* [online]. [cit. 2022-04-20]. Dostupné z: https://portswigger.net/web-security/access-control

[6] Apache HTTP Server. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2022-04-20]. Dostupné z: https://cs.wikipedia.org/wiki/Apache_HTTP_Server

[7] Broken Access Control. *OWASP* [online]. [cit. 2022-04-20]. Dostupné z: https://owasp.org/www-community/Broken_Access_Control

[8] BROOKS, Charles J., Christopher GROW, Philip CRAIG a Donald SHORT. *Cybersecurity essentials*. Indianapolis, Indiana: Sybex, John Wiley, 2018. ISBN 978-1-119-36239-5.

[9] Confidentiality, Integrity And Availability – The CIA Triad. *CertMike* [online]. [cit. 2022-04-20]. Dostupné z: https://www.certmike.com/confidentiality-integrity-and-availability-the-cia-triad

[10] Cross Site Scripting (XSS). *OWASP* [online]. [cit. 2022-04-20]. Dostupné z: https://owasp.org/www-community/attacks/xss/

[11] Cross Site Scripting (XSS). *OWASP* [online]. [cit. 2022-04-20]. Dostupné z: https://www.acunetix.com/websitesecurity/cross-site-scripting

[12] Directory traversal. *PortSwigger* [online]. [cit. 2022-04-20]. Dostupné z: https://portswigger.net/web-security/file-path-traversal

[13] How can I prevent SQL injection in PHP?. *Stack Overflow* [online]. 2022 [cit. 2022-04-20]. Dostupné z: https://stackoverflow.com/questions/60174/how-can-i-prevent-sql-injection-in-php

[14] Hypertext Markup Language. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2022-04-20]. Dostupné z: https://cs.wikipedia.org/wiki/Hypertext_Markup_Language

[15] Informace o webových aplikacích. *Adobe* [online]. 2021 [cit. 2022-04-20]. Dostupné z: https://helpx.adobe.com/cz/dreamweaver/using/web-applications.html

[16] JavaScript. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2022-04-20]. Dostupné z: https://cs.wikipedia.org/wiki/JavaScript

[17] KIPRIN, Borislav. Comprehensive Guide to Insufficient Logging & Monitoring and How to Prevent It. *Crashtest Security* [online]. 2021 [cit. 2022-04-20]. Dostupné z: https://crashtest-security.com/insufficient-logging-monitoring-guide/

[18] KIPRIN, Borislav. Using Components with Known Vulnerabilities and Stay Secure. *Crashtest Security* [online]. 2021 [cit. 2022-04-20]. Dostupné z: https://crashtest-security.com/using-components-with-known-vulnerabilities/

[19] KIPRIN, Borislav. Your Guide to Sensitive Data Exposure (Fuzzing) & How to Fix It. *Crashtest Security* [online]. 2021 [cit. 2022-04-20]. Dostupné z: https://crashtest-security.com/sensitive-data-exposure/

[20] KOLOUCH, Jan, Pavel BAŠTA a kol. *CyberSecurity*. Praha: CZ.NIC, z. s. p. o., 2019. ISBN 978-80-88168-34-8.

[21] Kybernetická bezpečnost: Typy útoků a prozíravý vývoj aplikací. *Pixman* [online]. [cit. 2022-04-20]. Dostupné z: https://www.pixman.cz/blog/kyberneticka-bezpecnost-typy-utoku-a-proziravy-vyvoj-aplikaci

[22] MySQL. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2022-04-20]. Dostupné z: https://cs.wikipedia.org/wiki/MySQL

[23] POZA, Diego. What Is Broken Authentication?. *Auth0* [online]. 2020 [cit. 2022-04-20]. Dostupné z: https://auth0.com/blog/what-is-broken-authentication/

[24] RADHAKRISHNAN, Reshmi. OWASP Top 10 : Security Misconfiguration. *Siemba* [online]. [cit. 2022-04-20]. Dostupné z: https://www.siemba.io/post/owasp-top-10-security-misconfiguration

[25] Sass (stylesheet language). In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2022-04-20]. Dostupné z: https://en.wikipedia.org/wiki/Sass_(stylesheet_language)

[26] SENSITIVE DATA EXPOSURE. *Contrast Security* [online]. [cit. 2022-04-20]. Dostupné z: https://www.contrastsecurity.com/knowledge-hub/glossary/sensitive-data-exposure

[27] SQL Injection. *OWASP* [online]. [cit. 2022-04-20]. Dostupné z: https://owasp.org/www-community/attacks/SQL_Injection

[28] STUTTARD Dafydd a Marcus PINTO. *The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws*. 2nd Edition. Indianopolis: John Wiley & Sons, Inc., 2021. ISBN 978-1118026472.

[29] ŠTRÁFELDA, Jan. PHP. *Strafelda* [online]. [cit. 2022-04-20]. Dostupné z: https://www.strafelda.cz/php

[30] Top 10 Security Risks in Web Applications. *GeeksforGeeks* [online]. 2021 [cit. 2022-04-20]. Dostupné z: https://www.geeksforgeeks.org/top-10-security-risks-in-web-applications/

[31] UPADHYAY, Isha. Broken Authentication: How to Prevent It, Examples and More. *Jigsaw Academy Education Pvt. Ltd.* [online]. 2020 [cit. 2022-04-20]. Dostupné z: https://www.jigsawacademy.com/blogs/cyber-security/broken-authentication/

[32] Web application (Web app). *TechTarget* [online]. 2021 [cit. 2022-04-20]. Dostupné z: https://www.techtarget.com/searchsoftwarequality/definition/Web-application-Web-app

[33] Web application security. *Digital* [online]. [cit. 2022-04-20]. Dostupné z: https://digital.ai/glossary/web-application-security