Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Bezpečnostní testy webové aplikace

Barbora Pilná

Bakalářská práce

2020

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: 2018/2019

ZADÁNÍ BAKALÁŘSKÉ PRÁCE
(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Barbora Pilná**
Osobní číslo:	**I16339**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Téma práce:	**Bezpečnostní testy webové aplikace**
Zadávající katedra:	**Katedra informačních technologií**

Zásady pro vypracování

Cílem BP je vytvořit testové úlohy zaměřené na zjištění slabých míst v zabezpečení webových aplikací. V teoretické části autor popíše: základní pojmy, hrozby, dopady, rizika, zranitelnosti, metody a postupy související s bezpečnostními testy. Praktická části BP bude aplikovat teoretické poznatky a pokryje následující oblasti:

- injektování
- chybná autentizace a správa relace
- Cross-Site Scripting (XSS)
- nezabezpečený přímý odkaz na objekt
- nezabezpečená konfigurace
- expozice citlivých dat
- chyby v řízení úrovní přístupů
- Cross-Site Request Forgery (CSRF)
- použití známých zranitelností komponent
- neošetřená přesměrování a předávání

Jednotlivé testy budou podrobně a návodně popsány a budou realizovány na vybrané webové aplikaci v právně bezpečném prostředí.

Rozsah pracovní zprávy: **40**
Rozsah grafických prací:
Forma zpracování bakalářské práce: **tištěná**

Seznam doporučené literatury:

SULLIVAN, Bryan a Vincent LIU. *Web application security: a beginner's guide*. New York: McGraw-Hill, c2012. ISBN 978-007-1776-165.
YATES, Anthony. *The Owasp Handbook – Everything You Need to Know about Owasp*. Emereo Publishing, 2016. ISBN 1489135227.
STUTTARD, Dafydd a Marcus PINTO. *The web application hacker's handbook: finding and exploiting security flaws*. 2nd ed. Chichester: John Wiley [distributor], c2011. ISBN 978-111-8026-472.

Vedoucí bakalářské práce: **Ing. Soňa Neradová, Ph.D.**
Katedra informačních technologií

Datum zadání bakalářské práce: **31. října 2018**
Termín odevzdání bakalářské práce: **12. května 2019**

Univerzita Pardubice Česká republika
L.S.

Ing. Zdeněk Němec, Ph.D.
děkan

Ing. Lukáš Čegan, Ph.D.
pověřený vedením katedry

V Pardubicích dne 14. prosince 2018

Prohlašuji:

Tuto práci jsem vypracovala samostatně. Veškeré literární prameny a informace, které jsem v práci využila, jsou uvedeny v seznamu použité literatury.

Byla jsem seznámena s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, 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. 9/2012, bude práce zveřejněna v Univerzitní knihovně a prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne 8. 8. 2020

Barbora Pilná

PODĚKOVÁNÍ

Ráda bych poděkovala Ing. Soně Neradové, Ph.D. za cenné rady, připomínky, věnovaný čas, trpělivost a ochotu v průběhu zpracování této práce. Také bych chtěla poděkovat svým kolegům za přínosné konzultace při psaní a svým rodičům za jejich podporu během celého studia.

ANOTACE

Cílem této práce je přiblížení bezpečnostních testů webových aplikací. V teoretické části jsou popsané základní koncepty, které jsou jádrem k pochopení podstaty bezpečnostních testů. Obsahuje vysvětlení pojmů jako hrozby, dopady, rizika nebo zranitelnosti. V práci je popsáno i dělení bezpečnostních testů a doporučené metody s nimi související.

Praktická část aplikuje a zužitkovává teoretické poznatky injektování, chybné autentizace a správy relace, Cross-Site Scripting (XSS), nezabezpečeného přímého odkazu na objekt, nezabezpečené konfigurace, expozice citlivých dat, chyb v řízení úrovní přístupů, Cross-Site Request Forgery (CSRF), použití známých zranitelností komponent, neošetřeného přesměrování a předávání.

KLÍČOVÁ SLOVA

Bezpečnost, OWASP, CSRF, XSS, webová aplikace, injektování, DVWA, OWASP Security Shepherd.

TITLE

Web application security tests.

ANNOTATION

This work deals with security tests of web technologies. The theoretical part describes basic concepts, threats, impacts, risks, vulnerabilities, methods and procedures related to safety tests. In the practical part, theoretical knowledge and proven injection methods, malicious authentication and session management, Cross-Site Scripting (XSS), unsecured direct object reference, unsecured configuration, sensitive data exposure, access control levels, CSRF, the use of known component vulnerabilities, untreated redirects and relaying.

KEYWORDS

Security, OWASP, CSRF, XSS, web application, injection, DVWA, OWASP Security Shepherd.

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM ZKRATEK

API	Application Programming Interface
CA	Certificate authority
CSRF	Cross-site request forgery
CVE	Common Vulnerabilities and Exposures
DNS	Domain Name Systém
DOM	Document Object Model
DVWA	Damn Vulnerable Web Application
ESAPI	Enterprise Security Application Programming Interface
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IP	Internet Protocol
LTS	Long-term support
MD5	Message-Digest algorithm
MITM	Man in the middle
OWASP	Open Web Application Security Project
PDF	Portable Document Format
SQL	Structured Query Language
SSH	Secure Shell
SSL	Secure Sockets Layer
TLS	Transport Layer Security
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
XSS	Cross-Site Scripting

ÚVOD

V posledních deseti letech se dvojnásobně zvýšil počet uživatelů internetu na celém světě a v České republice bylo v roce 2019 pokryto internetem 81 % domácností [1]. Tento fakt dopomohl většímu zájmu o webové aplikace a v dnešní době chytrých telefonů už je možné si prohlížet web i z mobilního telefonu. Díky tomu, že aktuálně tvoří návštěvnost internetu z telefonu 52 % z celkové návštěvnosti [2], si může webovou aplikaci prohlížet téměř kdokoli a kdykoli. S rozšířením webových aplikací ale vzrostl i zájem o jejich data a alarmující je skutečnost, že s narůstajícím počtem webových aplikací také úměrně narůstá počet těch, které jsou zranitelné.

Každá webová aplikace v současnosti sbírá data uživatelů a uchovává je, při tom právě informace jsou dnes velice cenným zbožím. Například personalizace reklam funguje právě na principu nasbíraných poznatků o daném uživateli a jeho preferencích a nutí tím uživatele ke koupi produktů, které v minulosti vyhledával nebo prohlížel. Webové aplikace ale mohou mít přístup k mnohem důležitějším datům, než je celé jméno, bydliště, e-mail, datum narození nebo číslo kreditní karty. Právě tyto informace je potřeba ochránit a zjistit slabá místa webové aplikace dříve, než je využije útočník. Zvláště této problematice se práce věnuje.

První kapitola práce se zabývá základními pojmy potřebnými pro pochopení vzniku bezpečnostních testů. Dále je uvedeno dělení testů a jejich úloha a metody a postupy, které s penetračními testy souvisí. Ke konci kapitoly je představen projekt Top Ten nadace Open Web Application Security Project (OWASP), která sdružuje elitu na bezpečnost. Kapitolu završuje tabulka hodnocení bezpečnostních rizik testovaných v práci podle OWASP klasifikace.

V druhé kapitole jsou charakterizovány aplikace a nástroje použité ke zpracování praktické části bakalářské práce. Jednotlivé zranitelnosti OWASP Top Ten jsou testovány na dvou aplikacích, a to na aplikaci Damn Vulnerable Web Application (DVWA) a OWASP Security Shepheard. Webové aplikace jsou testovány převážně manuálně a v některých případech je využito nástroje Burp Suite a pro automatické skenování je zvolena aplikace OpenVAS. Nástrojům je v kapitole věnován obecný popis a charakterizace modulů, které jsou v práci využity.

Třetí část práce objasňuje problematiku jednotlivých zranitelností. Za každou charakteristikou zranitelnosti následuje aplikování útoku na danou zranitelnost v aplikaci. Pokusy o napadení jsou realizovány na vybrané webové aplikaci podle možností procvičení probírané zranitelnosti. Na konci každé zranitelnosti jsou popsány metody obrany webové aplikace proti danému útoku.

1 ZÁKLADNÍ POJMY

1.1 Hrozby

Hrozba představuje nějakou událost, nebo entitu, která chce, nebo může poškodit společnost nebo její systém [3]. Má potenciál poškodit aktivum[1] společnosti jeho ztrátou, porušením důvěrnosti nebo nežádoucí změnou aktiva.

Tabulka 1: Příklady hrozeb

Člověk		Životní prostředí
Úmyslné	**Neúmyslné**	
Krádež	Nesprávné směrování	Požár
Odposlech	Detekce souborů	Zemětřesení
Modifikace informací	Fyzická nehoda	Záplavy
Zákeřný kód		Poškození bleskem

Zdroj: zpracováno dle [4]

1.2 Dopady

Dopad je výsledek nějaké chyby zabezpečení, která ovlivňuje aktivum. Mezi dopady je možné zařadit například poškození aktiva, informačního systému, kompromitace důvěrnosti, dostupnosti, integrity, autentičnosti nebo spolehlivosti [4]. Jako příklady nepřímých dopadů je možné uvést finanční ztráty, ztrátu pozice na trhu nebo ztrátu reputace společnosti. Posouzení dopadu je důležité při odhadu rizika.

1.3 Rizika

Riziko popisuje potenciál, že daná hrozba odhalí zranitelnost systému, nebo jeho části a způsobí škodu společnosti [3]. Na základě znalostí systému je možné odhadnout nebo stanovit jaké je riziko, že nastane daná hrozba. Riziko není možné úplně eliminovat. Vedení společnosti tedy musí akceptovat zbytkové riziko a být schopno přijmout případný dopad.

V případě zaměření se na metodiku hodnocení rizik, je doporučený následující postup. Nejprve dochází ke zhodnocení aktiv společnosti, která jsou ohodnocena podle jejich důležitosti.

[1] Aktivum je celkový (i nehmotný) majetek společnosti.

Následně je provedena klasifikace dopadů na společnost při jakémkoli narušení aktiv a je vynásobena pravděpodobnost, že daný stav nastane, a jeho dopad [4]. Je žádoucí zjistit, zda je riziko akceptovatelné, nebo vyžaduje řešení.

1.4 Zranitelnosti

Zranitelnost je slabina aktiva, která může být využita některou z hrozeb a způsobit tím škodu infrastruktury systému nebo obchodním cílům [4]. Zranitelnost není vázána na hrozbu, může se tedy vyskytnout i bez existence odpovídající hrozby. Sama o sobě tedy nezpůsobuje žádnou škodu, pouze poskytuje prostor hrozbě, která může následně jednat a využít tuto zranitelnost k poškození aktiv společnosti. Zranitelnost může být bezcenná, pokud se aktivum změní tak, že zranitelnost nadále nelze aplikovat. Zranitelnost by měla být posuzována jednotlivě i souhrnně, aby se zohlednil plný provozní kontext společnosti. Protože při každé změně systému se mohou změnit i jeho zranitelnosti, měl by být vždy prověřen. Důležité je včas provést nápravná opatření systému proti jeho zneužití.

1.5 Druhy bezpečnostních testů

Testování webových aplikací se využívá ke zjištění a eliminaci chyb již ve stádiu vývoje příslušné aplikace. Žádná testovací metoda, nebo jejich kombinace, nedokáže odhalit všechny zranitelné body v kódu. Metody bezpečnostních testů lze rozdělit do dvou hlavních kategorií, kde každá z nich obsahuje vlastní podkategorie. Dělí se podle způsobu provedení, kam lze zařadit manuální testy, automatizované testy a semiautomatické testy. Druhá kategorie se dělí podle znalostí o testovaném systému, do kterých spadají black-box, white-box a grey-box testování. [5, s. 14-16]

1.5.1 Manuální testy

Hovoříme o manuálním testování, pokud je prováděno testerem. Při manuálním testování je výhoda vytvoření testů na míru pro daný systém, použití automatického testování v případě vývoje nové aplikace, se nemusí podařit odhalit všechny nefunkčnosti. Další výhodou je, že tester zvládne interpretovat výsledky testování i osobám, které se v dané oblasti neorientují a dané problematice nerozumí. Sám ví, co a jak testuje a proč zvolil daný postup, proto je schopen výsledky objasnit například vedení společnosti nebo investorům. [5, s. 16]

Nevýhodou manuálního testování je především časová a znalostní náročnost. Vzhledem k obsáhlým technologiím, je potřeba aby tester disponoval zevrubnými vědomostmi o tvorbě webových aplikací. [5, s. 16]

1.5.2 Automatizované testy

Tyto testy mají výhodu v rychlosti a reprodukovatelnosti. Tester se musí naučit ovládat nástroj, který byl vytvořen profesionálem ve svém oboru, a proto se zkrátí čas testování. [5, s. 16] Je jednodušší se naučit nástroj ovládat než pochopit, jak test funguje, proč je zvolen daný postup a co přesně je potřeba udělat, než jak tomu je u manuálního testování.

Automatizovaným testům většinou tester dokonale nerozumí, a proto není reálné očekávat, že bude schopen podrobnosti dané problematiky vysvětlit nezainteresované osobě. Při automatizovaných testech nelze testovat některá specifická zranitelná místa, tato skutečnost je samotnou nevýhodou toho typu testování. Tyto testy jednoduše nemohou pokrýt zvláštnosti testovaného systému, jsou spíše obecné. [5, s. 16]

1.5.3 Semiautomatické testy

Semiautomatické testy představují mezistupeň mezi manuálními a automatickými testy. Vytvářejí jejich kombinaci za účelem využití rychlosti automatických testů a možnosti otestování systému na specifické zranitelnosti. Testy se snaží eliminovat nevýhody a uvést v praxi výhody obou předešlých testů. [5, s. 16]

1.5.4 Black-box testy

Tento druh testů se v češtině nazývá testování černé skříňky, protože tester nezná funkcionalitu systému, který testuje. Jedná se o simulaci vnějšího přístupu útočníka bez znalosti vnitřní struktury aplikace, nebo sítě. Tester je obeznámen pouze se základním účelem aplikace, takže černou skříňku představuje zdrojový kód, ke kterému nemá přístup. [5, s. 17]

S přihlédnutím ke zmíněnému, lze považovat za výhodu absence znalosti konkrétního programovacího jazyka a použitých technologií. Další výhodou je možnost přizpůsobení testů podle požadavků zadavatele.

Naopak nevýhodou je potřeba širokých znalostí testera. V některých případech je potřeba použití sofistikovanějších přístupů a chybí kontrola efektivity kódu. [5, s. 17]

1.5.5 White-box testy

Na rozdíl od předchozí metody testování, má tester přístup ke zdrojovému kódu a zná architekturu testované aplikace. Tester musí ovládat daný programovací jazyk, ve kterém je kód napsaný. Tento kód by měl být okomentovaný a korektně napsaný, aby se v něm mohl kdokoli, kdo ovládá daný jazyk, rychle zorientovat a vyznat. [5, s. 17]

Hlavní výhodou je rychlost nalezení zranitelnosti i v případě, že se provede podrobnější kompletní analýza. Dále je možné optimalizovat kód na základě nalezených zranitelných míst a chyb.

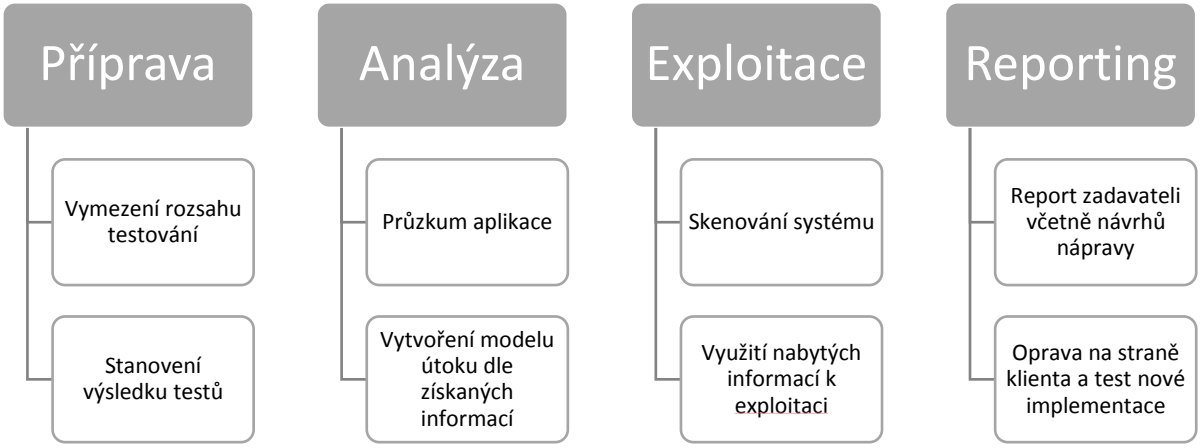
U tohoto typu testování nutná znalost programovacího jazyka, ve kterém je aplikace napsaná. Tato skutečnost může zvýšit cenu testu, protože na testera jsou kladeny vyšší nároky v podobě užšího zaměření jak na kód, tak na architekturu dané aplikace. [5, s. 17]

1.5.6 Grey-box testy

Grey-box testy představují alternativu k předchozím dvěma typům testů. Snaží se zkombinovat a využít výhody jak black-box, tak i white-box testů. Během testování se využívá znalosti vnitřní logiky a struktury aplikace, ale samotné testy probíhají z pohledu potenciálního útočníka. Tyto testy také mohou obsahovat metody reverzního inženýrství za účelem stanovení limitních hodnot vstupních údajů aplikace. [5, s. 17]

1.6 Metody a postupy související s bezpečnostními testy

Obecné metodiky penetračního testování se liší podle zdroje, nejsou jednotné, ale základní struktura zůstává podobná. Postupy testů jsou upraveny a implementovány podle vlastních zkušeností a požadavků. Díky této skutečnosti je doporučených metod a postupů velké množství a z toho důvodu bude v této práci zmíněna pouze základní struktura. Postup bezpečnostního testování je rozdělen do čtyř fází, které jsou popsány v následujících podkapitolách. [5, s. 11-26]



Obrázek 1: Fáze penetračního testování

Zdroj: vlastní

1.6.1 Stanovení cílů a rozsahu penetračních testů

Toto je první fáze postupu. V tomto bodě je potřeba stanovit prioritní cíle a rozsahy, aby je bylo možné rozložit na jednotlivé etapy, které se budou testovat. Je potřeba mít na paměti, že není možné cokoli ověřit stoprocentně. [5, s. 11]

Pro bližší specifikaci je uveden následující příklad: je přijat požadavek na otestování dané webové aplikace a z něj se musí určit, co se musí otestovat prioritně. Může to být bezpečnost přihlašování nebo bezpečnost transakcí uživatelů. Dalším příkladem by mohlo být ověření stability aplikace nebo například ověření zabezpečení přístupu ke konfiguraci aplikace. Dalším fatálním problémem by mohl být přístup k informacím o uživatelích, potažmo do databáze webové aplikace. Jak je vidět možností, jak rozebrat primární požadavek je mnoho, proto je tento bod nezbytný, aby bylo testováno to, co si zadavatel opravdu přeje. [5, s. 11-26]

1.6.2 Průzkum a získávání informací

Podle vyhodnocení v první fázi je potřeba rozhodnout jaký druh bezpečnostních testů bude použit, jejich výhody a nevýhody již byly popsány v podkapitole 1.5. V této fázi je nutné získat představu jak a kde hledat informace o dané aplikaci. Tyto informace se poté použijí coby vstup v další fázi penetračního testování. [4]

Jako příklad je možné uvést potřeba otestovat bezpečnost webové aplikace. V této fázi lze aktivně skenovat adresářovou strukturu ze vzdáleného počítače za účelem odhalení struktury adresářů serveru, kam aplikace ukládá svoje data. Lze tedy například zjistit, jaké složky obsa-

hují informace o uživatelských účtech. K tomuto účelu se využívá aplikace, která dokáže vytvořit tzv. zrcadlovou kopii serveru, tu je následně možné prohlížet a analyzovat získané informace. Do sběru dat můžeme zahrnout informace: o typu používaných zařízení, verzích operačních systémů nebo používaného softwaru. [5]

1.6.3 Skenování a exploitace

Ve třetí fázi probíhá obecné skenování testovaného systému a testování jeho zabezpečení. Bezpečnost se testuje pokusy o prolomení bezpečnostních mechanizmů celého systému. Jedním z cílů exploitace[2] by mohl být přístup do systému nebo databáze bez zadání validních přihlašovacích údajů. Potom, co útočník získá přístup do databáze nebo systému, získá citlivé informace, a dokonce může poškodit aktiva společnosti úpravou nebo úplným smazáním dat. V té samé době útočník disponuje možností úplného znepřístupnění služby, což může mít za následek některý z dopadů popsaných dříve. [5]

V dnešní době je vývoj poznatků a znalostí opravdu rychlý, proto skutečnost, že ještě nebylo detekováno prolomení určitého bezpečnostního mechanismu, neznamená, že k jeho prolomení nedojde v budoucnu [6]. Celá tato fáze je postavená na využívání chyb a nedostatků v systémech a nezáleží, zda se jedná o dobře známé chyby, nebo teprve nedávno objevené.

Skenování a exploitace může využívat nespočetné množství postupů, nástrojů a testovacích aplikací. Mezi základní útoky lze zařadit například ruční injekci parametrů, odposlech obsahu nebo analýzu a útok na token relace [7].

1.6.4 Report

Účelem závěrečné zprávy je shrnutí výsledků jednotlivých testů a poznatků získaných při testování [5]. Při psaní reportu je nutné dbát na srozumitelnost, aby byl programátor schopen bez dalších upřesňujících informací zranitelnost opravit. Zpráva by měla zahrnovat: [8]

- název chyby daného zabezpečení,
- zranitelný koncový bod,
- technický popis zranitelnosti,
- obchodní dopady a závažnost těchto dopadů,
- video nebo obrázek jako důkaz (tzv. Proof of Concept).

[2] Exploitace je využití chyb v programu za účelem nekalých praktik.

1.7 OWASP

Open Web Application Security Project je nezisková nadace, která se snaží o zlepšení softwarové bezpečnosti. Nadace byla založena 21. dubna 2004 ve Spojených státech amerických a má desítky tisíc členů a pořádá vzdělávací konference. Pod nadací OWASP existuje více než 150 různých open-source projektů, bezpečnostních nástrojů, standardů a technických materiálů. Jedná se o otevřenou komunitu odborníků z celého světa. [9]

1.7.1 OWASP Top Ten

Jeden ze standardů pod záštitou nadace je OWASP Top Ten, který obsahuje deset nejčastějších bezpečnostních rizik webových aplikací. Většinou je vydávána každé tři roky kvůli aktualizaci zranitelností. Na začátku je vždy porovnání s předchozí verzí, následované krátkým přehledem všech deseti chyb, doplněných výstižnou charakteristikou. Dále jsou zde uvedeny jednotlivé hrozby a jejich podrobnější popis. Každé riziko je hodnoceno podle následující tabulky:

Tabulka 2: Identifikace rizik dle OWASP Top 10

Původci hrozby	Vektory útoku	Rozšíření slabiny	Zjistitelnost slabiny	Technické dopady	Obchodní dopady
Specifické pro aplikaci	Snadný	Rozsáhlé	Snadná	Vážný	Specifické pro aplikaci a podnikání
	Průměrný	Běžné	Průměrná	Střední	
	Obtížný	Vzácné	Obtížná	Malý	

Zdroj: zpracováno dle [10]

Následuje odstavec, který zabývá zjištěním, zda je aplikace ohrožena danou zranitelností. Je zde například uvedeno, jak se v aplikaci zachází s daty, aby došlo k této zranitelnosti a na co si dát obecně pozor. Další odstavec popisuje, co implementovat, aby bylo možné zranitelnosti předejít. Jsou zde uvedeny i příklady scénáře útoku s jejich krátkým popisem. Nakonec každá zranitelnost obsahuje odkazy na různé další zdroje souvisejícími s danou zranitelností. [10; 11]

V této práci je uveden rozbor nejkritičtějších hrozeb podle OWASP Top Ten, kde se často jedná a nedostatečné zabezpečení, chybnou implementaci, použití známých zranitelných komponent nebo například neošetření přesměrování. Všechna rizika, popsaná v praktické části této práce, jsou shrnuty do tabulky uvedené níže. Je zde uvedena složitost zneužití, jak rozšířená

je daná zranitelnost, také jak těžké je její odhalení a jaký je technický dopad v případě, že dojde ke zneužití.

Tabulka 3: Přehled rizik uvedených v práci a jejich charakteristika dle OWASP

Riziko	Zneužitelnost	Rozšíření	Zjistitelnost	Dopad
Injektování	Snadná	Běžné	Průměrná	Vážný
Chybná autentizace	Průměrná	Rozsáhlé	Průměrná	Vážný
XXS	Průměrná	Velmi rozsáhlé	Snadná	Střední
Nezabezpečený odkaz	Snadná	Běžné	Snadná	Střední
Konfigurace	Snadná	Běžné	Snadná	Střední
Citlivá data	Obtížná	Vzácné	Průměrná	Vážný
Řízení přístupu	Snadná	Běžné	Průměrná	Střední
CSRF	Průměrná	Běžné	Snadná	Střední
Komponenty	Průměrná	Rozsáhlé	Obtížná	Střední
Přesměrování	Průměrná	Vzácné	Snadná	Střední

Zdroj: zpracováno dle [10]

2 POUŽITÉ NÁSTROJE A APLIKACE

Cílem této kapitoly je představení nástrojů a testovacích webových aplikací, používaných v této práci. Kapitola popisuje jejich vlastnosti a možnosti. Testovací aplikace byly zvoleny tak, aby byly stále aktuální, udržované a pokryly cíle práce. Použité nástroje pro ulehčení testování byly vybrány takové, které jsou hojně používané v komunitě penetračních testerů a odpovídají potřebám testování webové aplikace v následující kapitole.

Nastavení a orientace v použitých aplikacích je samonávodné a lehce pochopitelné. Obě aplikace mají k dispozici menu po levé straně, které obsahuje názvy modulů. Každý modul disponuje právě tou zranitelností, kterou je pojmenován. Veškerá praktická část prováděná v této práci je označena přesným a úplným názvem modulu uvedeným v závorkách a popis je doplněn i o příslušnou aplikaci. Nadpis každé praktické části dané zranitelnosti tedy vypadá následovně: Praxe Název testovací aplikace (Název modulu příslušné aplikace).

2.1 Damn Vulnerable Web Application

DVWA je záměrně zranitelná webová aplikace založená na PHP a MySQL. Jejím hlavním cílem je pomáhat vývojářům lépe porozumět procesům zabezpečení webových aplikací a pomoci bezpečnostním testerům zdokonalit se v jejich dosavadních dovednostech v právně bezpečném prostředí [12]. Aplikace umožňuje nastavení více úrovní zabezpečení, pokud není napsáno jinak, testy prováděné na této aplikaci byly nastaveny na *Low*. Na obrázku níže je možné vidět na jaké druhy testů je aplikace zaměřena.

Obrázek 2: Nastavení zabezpečení v aplikaci DVWA

Zdroj: vlastní

Instalace a nastavení aplikace je podrobně popsána na vlastním[3] GitHubu, včetně nastavení databáze a potřebných komponent. Další možností je spustit DVWA přes Docker a i tato varianta je na GitHubu uvedena.

2.2 OWASP Security Shepherd

OWASP Shepherd je webová a mobilní aplikační bezpečnostní platforma. Byla navržena tak, aby podporovala a zlepšovala povědomí o bezpečnosti u různých zranitelností. Cílem tohoto projektu je oslovit bezpečnostní nováčky nebo zkušené inženýry a prohloubit jejich dovednosti testování penetrace na stav bezpečnostního experta [13]. Instalace aplikace je opět popsána i s jejími potřebnými komponentami na GitHubu této[4] aplikace.

[3] https://github.com/ethicalhack3r/DVWA
[4] https://github.com/OWASP/SecurityShepherd

Aplikace prezentuje koncepty bezpečnostních rizik uživatelům v lekcích následovaných výzvami. Lekce poskytuje laickou nápovědu o konkrétním bezpečnostním riziku a pomáhá využít další materiály o tomto problému. Výzvy zahrnují slabou bezpečnost zranitelností, které uživatelům ponechávají prostor pro jejich zneužití.

Obrázek 3: Vysvětlení problematiky lekce v aplikaci OWASP Shepherd

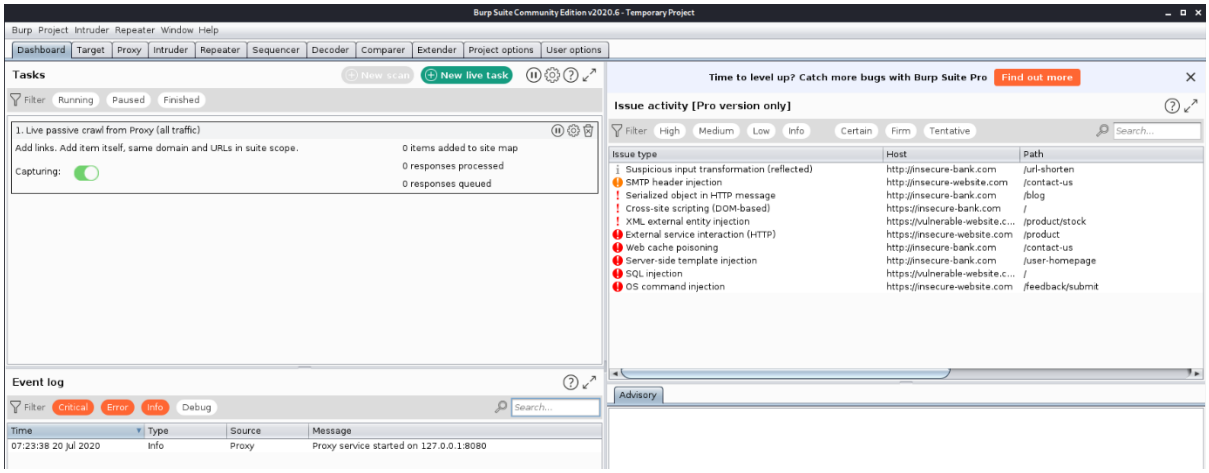
Zdroj: vlastní

2.3 Burp Suite

Burp Suite obsahuje řadu funkcí a možností ručního a automatizovaného testování bezpečnosti. Je distribuován ve třech verzích. Bezplatná *Community* edice, která obsahuje pouze základní manuální nástroje potřebné pro testování. Placená verze *Professional* nabízí podporu skeneru zranitelností webových aplikací, pokročilé a základní manuální nástroje určené k testování. Cena této verze se pohybuje kolem 349 USD za rok. Poslední verzí je *Enterprise*, která se zaměřuje na automatizaci testů a postrádá tedy manuální nástroje. Tato verze může nabídnout: skener zranitelností webových aplikací, plánování a periodické spouštění testů, paralelní skenování neurčeného množství stránek zároveň a integraci testů se současným použitím verzovacího systému, a to ještě před vypuštěním nové funkcionality do produkce. Cenově se verze pohybuje kolem 3 499 USD za rok. [14]

Použité moduly aplikace Burp Suite Community edice v této práci: [15]

- ***Sequencer*** – Analyzuje kvalitu náhodnosti bezpečnostních tokenů relací. Lze využít k testování kvality session tokenů, ochranu formulářů před útokem CSRF, nebo jiných důležitých datových položek, které mají být nepředvídatelné. Výsledky testů jsou graficky zpracovány a náhodnost je vyjádřena i slovně.
- ***Proxy*** – Tento modul zachytává komunikaci jakožto prostředník mezi prohlížečem a serverem. Umožňuje tak zobrazit a upravit všechny požadavky a odpovědi předávané mezi prohlížečem a cílovými webovými servery. Pro správnou funkci tohoto modulu je potřeba stáhnout Burp Suite CA certifikát a přidat ho mezi autorizované ve webovém prohlížeči.

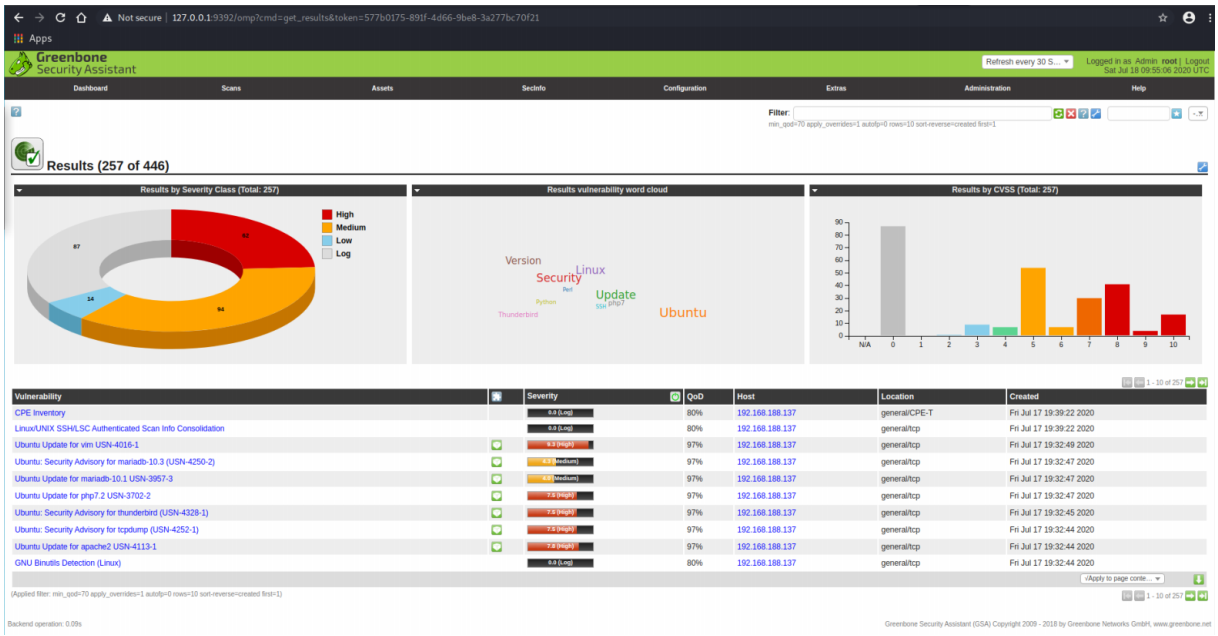


Obrázek 4: Přehled aplikace Burp Suite

Zdroj: vlastní

2.4 OpenVAS

OpenVAS je pokročilý skener zranitelností. Mezi jeho možnosti patří skenování různých internetových a průmyslových protokolů, vyladění výkonu pro skenování a interní programovací jazyk pro provádění jakéhokoli typu testu zranitelnosti. Skener je doplněn zdrojem více než 50 000 testů zranitelností s denními aktualizacemi. [16]



Obrázek 5: Výstup skenování z webového rozhraní aplikace OpenVAS

Zdroj: vlastní

Výběr komponent z kompletního soupisu[5], které jsou součástí skeneru OpenVAS:

- **Nmap** – Jedná se o bezpečnostní skener portů.
- **Nikto** – Zjišťuje, zda se na webovém serveru nevyskytují nebezpečné skripty, zastaralý software a jiné problémy. Provádí obecné i specializované testy.
- **Acunetix** – Webový bezpečnostní skener, který nabízí náhled do zabezpečení organizace ze všech možných úhlů.
- **Debian Security Analyzer** – Analyzuje seznam nainstalovaných balíčků a hlásí zranitelnosti nalezené v systému.
- **W3af** – Otevřený bezpečnostní skener webových aplikací. Nabízí skener zranitelností a exploitační nástroje pro webové aplikace. Získává informace o bezpečnostních zranitelnostech a poskytuje rady pro testování průniku do aplikace.

[5] https://wald.intevation.org/frs/?group_id=29

3 ROZBOR ZRANITELNOSTÍ

V této části práce jsou postupně popsány nejčastější zranitelnosti webové aplikace. Nejprve je vysvětleno, jak zranitelnost funguje, jaké jsou její druhy, pokud se dělí, a co by mohla způsobit. Po objasnění problematiky je popsaná praktická část, která využívá dané zranitelnosti na aplikaci DVWA, nebo Security Shepherd. Tato část je návodně popsána a doplněna o obrázky pro lepší ozřejmení postupu. Poslední část věnovaná dané zranitelnosti obsahuje postupy a návrhy pro její eliminaci.

3.1 SQL Injection

Díky této metodě, může útočník odeslat nedůvěryhodná data do interpretu jako součást původního příkazu nebo dotazu a tím získat přístup k datovému úložišti. Zde jsou většinou uloženy důležité informace jako povolení přístupu, citlivé informace o uživatelských účtech, nebo například konfigurace aplikace [7]. Útočník může ovlivnit interakci aplikace s úložištěm tak, aby byl schopen načíst, nebo upravit různá data. Mezi nejběžnější úložiště dat patří SQL databáze a LDAP úložiště. SQL injekce se dělí do tří hlavních kategorií: [17]

1. **In-Band** – útočník používá stejný komunikační kanál jak pro injektování kódu, tak i pro získání dat. Získaná data jsou tedy zobrazena přímo na webové stránce.
2. **Out-of-Band** – útočník není schopen použít stejný kanál pro injekci a získání dat. Spoléhá se na dostupnou funkci vytvoření DNS nebo HTTP kanálu ze strany databáze pro zobrazení získaných dat.
3. **Inferential** – nedochází k žádnému skutečnému přenosu dat. Princip spočívá v dotazování se databáze na pravda / nepravda otázky a porovnávání rozdílů mezi odpověďmi. Z toho důvodu se tento způsob označuje jako „blind“ neboli naslepo.

Klíč k injekci SQL databáze je „Structured Query Language“ neboli SQL. Tento jazyk může být použit pro čtení dat, aktualizaci, přidávání nových dat a jejich smazání z databáze. SQL je interpretovaný jazyk a webové aplikace běžně používají příkazy SQL, které obsahují data dodaná uživatelem. Pokud se špatně implementuje interpret pro komunikaci mezi aplikací a databází, aplikace může být zranitelná vůči SQL injekci [7]. Jedná se o jednu z nejznámějších zranitelností webových aplikací. V nejzávažnějších případech může tento druh injekce umožnit anonymnímu útočníkovi číst a upravit všechna data uložená v databázi, nebo dokonce převzít plnou kontrolu nad serverem, na kterém databáze běží.

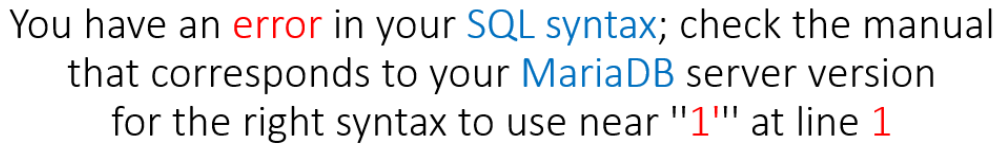
S vývojem povědomí o bezpečnosti webových aplikací se tato zranitelnost stala méně rozšířenou a je obtížnější ji detekovat a využít. V dnešní době už se aplikace brání SQL injekci

pomocí API, které je při správném použití bezpečné proti tomuto útoku. Hledání této zranitelnosti tedy může být velmi obtížný úkol, který vyžaduje dávku vytrvalosti k nalezení hrstky případů v aplikaci.

V mnoha situacích se doporučuje mít přístup ke stejné verzi databáze, kterou používá cílová aplikace [10]. Důvodem může být potřeba odladění části syntaxe, nebo nutnost nahlédnutí do tabulky či funkce. Odpovědi získané od dané aplikace totiž mohou být často neúplné nebo zašifrované, což může vyžadovat jistou dávku dedukce. V případě přístupu k této pracovní verzi databáze dojde ke značnému zjednodušení a potřeba dedukce se redukuje.

3.1.1 Praxe DVWA (*SQL Injection*)

Pokud je po zadání příkazu *1'* do vyhledávacího pole zobrazena hláška, která znamená, že aplikace je náchylná na SQL injekci. Je tedy možné do pole zadávat SQL příkazy.



Obrázek 6: Chybová zpráva databáze potvrzující možnost SQL injekce

Zdroj: vlastní

Po zadání kódu *1' OR 1=1#* je v aplikaci sestaven dotaz *SELECT * FROM users WHERE id='1' OR '1'='1 ',* kde znak # v kódu říká, aby se zbytek sestaveného dotazu ignoroval. Tento dotaz tedy zobrazí všechny uživatele z tabulky users, protože klauzule where se vždy vyhodnotí true.

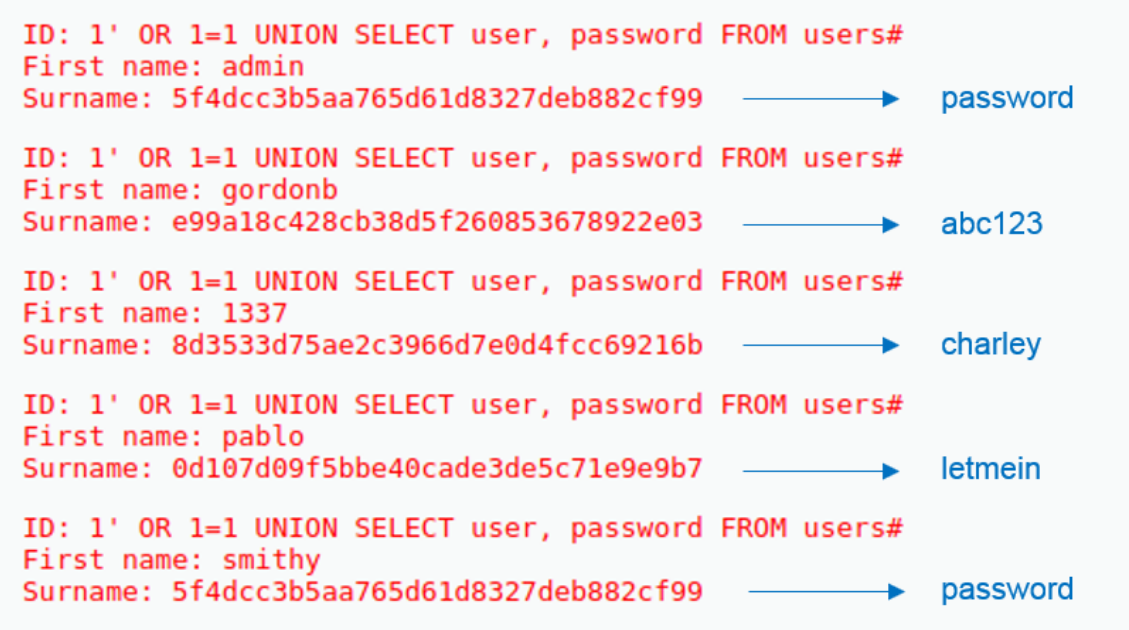
Obrázek 7: Výstup DVWA při získání všech uživatelů z databáze

Zdroj: vlastní

Díky předchozímu výstupu vyhledávání bylo zjištěno, že se odpovědi dělí na dva sloupce, takže například v tom druhém, si lze zobrazit názvy všech tabulek, které aplikace obsahuje. Do pole stačí zadat následující příkaz: *1' OR 1=1 UNION SELECT NULL, TABLE_NAME FROM INFORMATION_SCHEMA.TABLES#.* Systémová tabulka INFORMATION_SCHEMA.TABLES existuje v každé aplikaci, takže s její pomocí je možné si zobrazit názvy tabulek. Ty se s pomocí kódu výše zobrazí v druhém sloupci výstupu aplikace, který je definován jako TABLE_NAME.

Nyní, když jsou útočníkovi známy názvy tabulek, může se pokusit získat kontrolu nad systémem tím, že ukradne přihlašovací údaje administrátorovi webové aplikace. Toho se dá dosáhnout použitím kódu *1' OR 1=1 UNION SELECT user, password FROM users#,* který je zadán do vyhledávacího pole. Výsledkem je zobrazení přihlašovacích údajů všech uživatelů. Pokud aplikace používá slabé, zastaralé nebo dokonce žádné šifrování hesla útočníkovi stačí na jeho dešifrování použít například online nástroj. Výsledek dotazu, na kterém jsou zelenou barvou doplněná hesla, která byla dešifrovaná s pomocí toho[6] online nástroje. V tomto případě byla hesla šifrována s pomocí hashovací funkce MD5.

[6] https://www.md5online.org/md5-decrypt.html



Obrázek 8: Dešifrované přihlašovací údaje získané SQL injekcí

Zdroj: vlastní

3.1.2 Obrana

Jedním z druhů obrany je implementace předpřipravených příkazů, které zajišťují, že útočník není schopen změnit dotaz. Je potřeba si z pole vždy načíst parametr a ten celý dosadit do předpřipraveného SQL dotazu na databázi. Příklad této části je uveden na následujícím obrázku v pseudokódu:

```
1 // Načtení obsahu pole do proměnné id
2 String id = request.getParameter("User ID");
3 String dotaz = "SELECT * FROM user WHERE user_id = ?";
4 // Do předchozího dotazu se nastaví za otazník proměnná id
5 dotaz.setParameter(1, id);
6 // Následuje provedení dotazu
```

Obrázek 9: Pseudokód obrany před SQL injekcí předpřipravenými příkazy

Zdroj: vlastní

Podobným řešením je implementace uložených procedur [18]. Hlavní myšlenka spočívá v tom, že SQL dotaz pro proceduru je definován a uložen v samostatné databázi a je zavolán

z webové aplikace. Protože jsou tyto dvě metody velmi podobné, jsou stejně účinné při předcházení SQL injekci. Uložené procedury však nesmí obsahovat dynamické generování SQL, které je v tomto případě nebezpečné.

Další možností je únik všech vstupů zadaných uživatelem. S pomocí této metody je umožněno uniknout vložení vstupu uživatele do SQL dotazu. Tato technika by měla být zvolena pouze jako poslední možnost [18]. Výše zmíněné ověření vstupů je vhodnější volbou, protože ve srovnání s jinými druhy obrany nelze u této metody zaručit, že zabrání veškeré SQL injekci. Občas se doporučuje, pro zastaralý kód, kde je implementace metody ověření vstupu finančně nákladné.

3.2 Chybná autentizace a správa relace

Autentizace patří mezi nejjednodušší ze všech bezpečnostních mechanismů používaných ve webových aplikacích. V typickém případě uživatel zadá své uživatelské jméno a heslo a aplikace poté musí ověřit, zda jsou tyto informace správné. Pokud ano, požadovanou operaci uživateli umožní, pokud ne, operaci zakáže. Jedná se o přední linii obrany proti neoprávněnému přístupu. Pokud je útočník schopen tuto obranu prorazit, většinou získá plnou kontrolu nad aplikací včetně neomezeného přístupu k jejím údajům [7]. Bez správně fungující autentizace nemůže být žádný z ostatních základních bezpečnostních mechanismů, jako je například správa relací, účinný.

Útočník může využít trhliny v autentizaci či ve funkcích správy relace k tomu, aby se vydával za uživatele. Přestože správné vytvoření vlastního řešení autentizace a schémat správy relace je těžké, vývojáři často vytvářejí své vlastní. V důsledku toho, mají tato vlastní řešení často nedostatky v odhlášení, správě hesel, době platnosti, zapamatování si přihlášení, aktualizacích účtu a podobných oblastech. Nalezení takových nedostatků někdy může být obtížné, protože každá implementace je jedinečná. Tyto chyby však mohou umožnit napadení některých, či dokonce všech účtů. Když se útok podaří, útočník může provádět vše, co může dělat oběť. Častým cílem proto bývají privilegované účty.

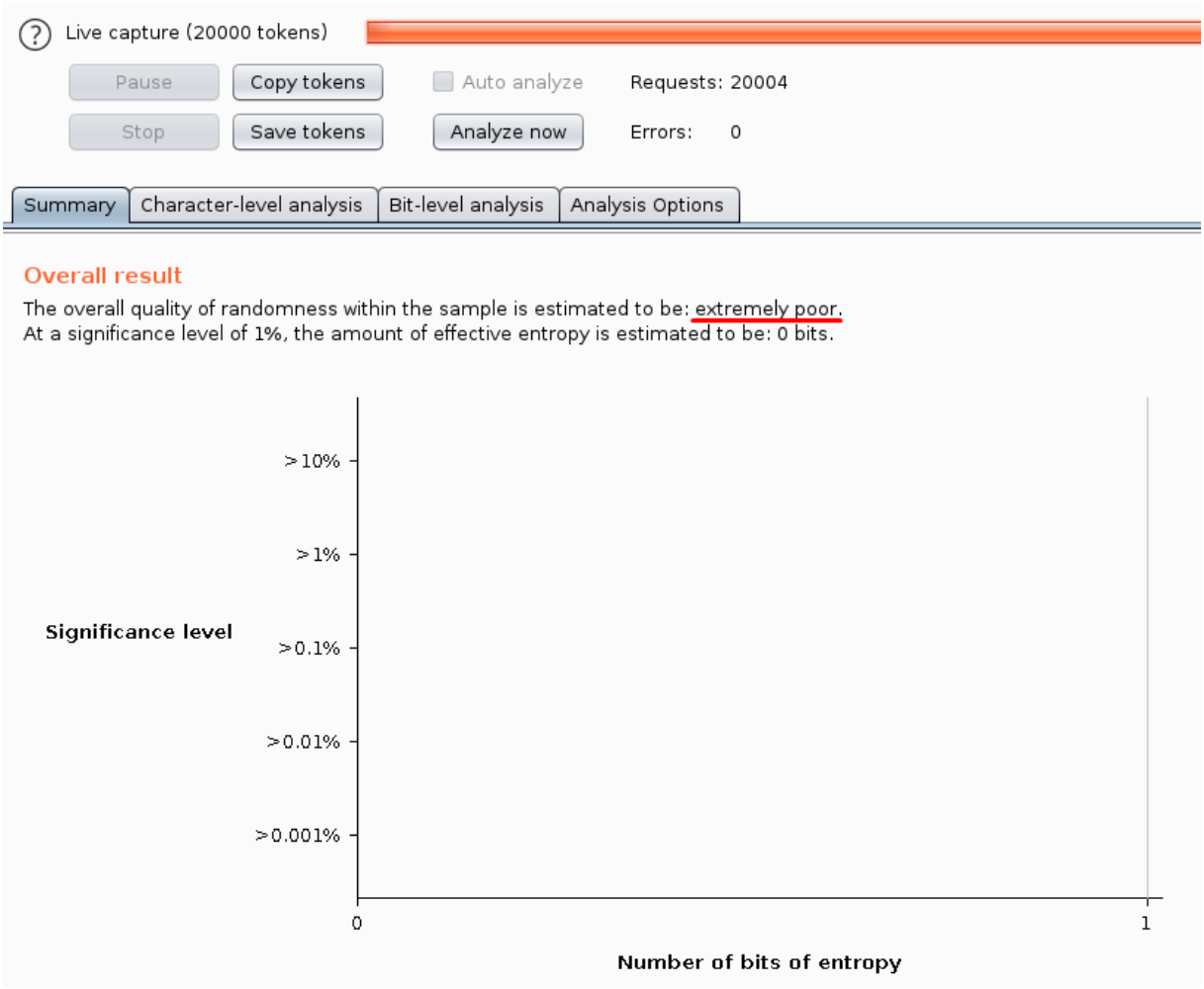
Jednou z nejčastějších zranitelností při ověřování jsou uživatelé a jejich volba hesla. Ve snaze toto heslo uhodnout může kdokoli do přihlašovacího formuláře napsat slova ze slovníku neboli provést slovníkový útok. V jiných případech se mohou drobné vady skrývat hluboko v aplikaci, které lze odhalit a zneužít až po pečlivé analýze složitého vícestupňového přihlašovacího mechanismu. Dalo by se tedy předpokládat, že vícestupňové přihlašovací mechanismy jsou méně náchylné na problémy s bezpečností oproti standardnímu ověřování

uživatelským jménem a heslem. To ale nemusí vždy platit. Provedení několika kontrol ověřování může značně přidat na bezpečnosti mechanismu, ale znamená to větší množství kódu, ve kterém se mohou vyskytnout další zranitelnosti [10]. V některých případech, může toto řešení vést k méně bezpečnému přihlašování, než je to na základě uživatelského jména a hesla.

Nejčastějším útokem za účelem získání přístupu do aplikace je útok hrubou silou neboli Brute-Force attack [10]. Jedná se o nejjednodušší formu útoku pro prolomení autentizace. Pokouší se zadat správně zadat uživatelské jméno a jeho heslo a zkouší to do té doby, dokud se mu nepovede přihlásit. Tato metoda bývá často automatizovaná.

3.2.1 Praxe DVWA (*Weak Session IDs*)

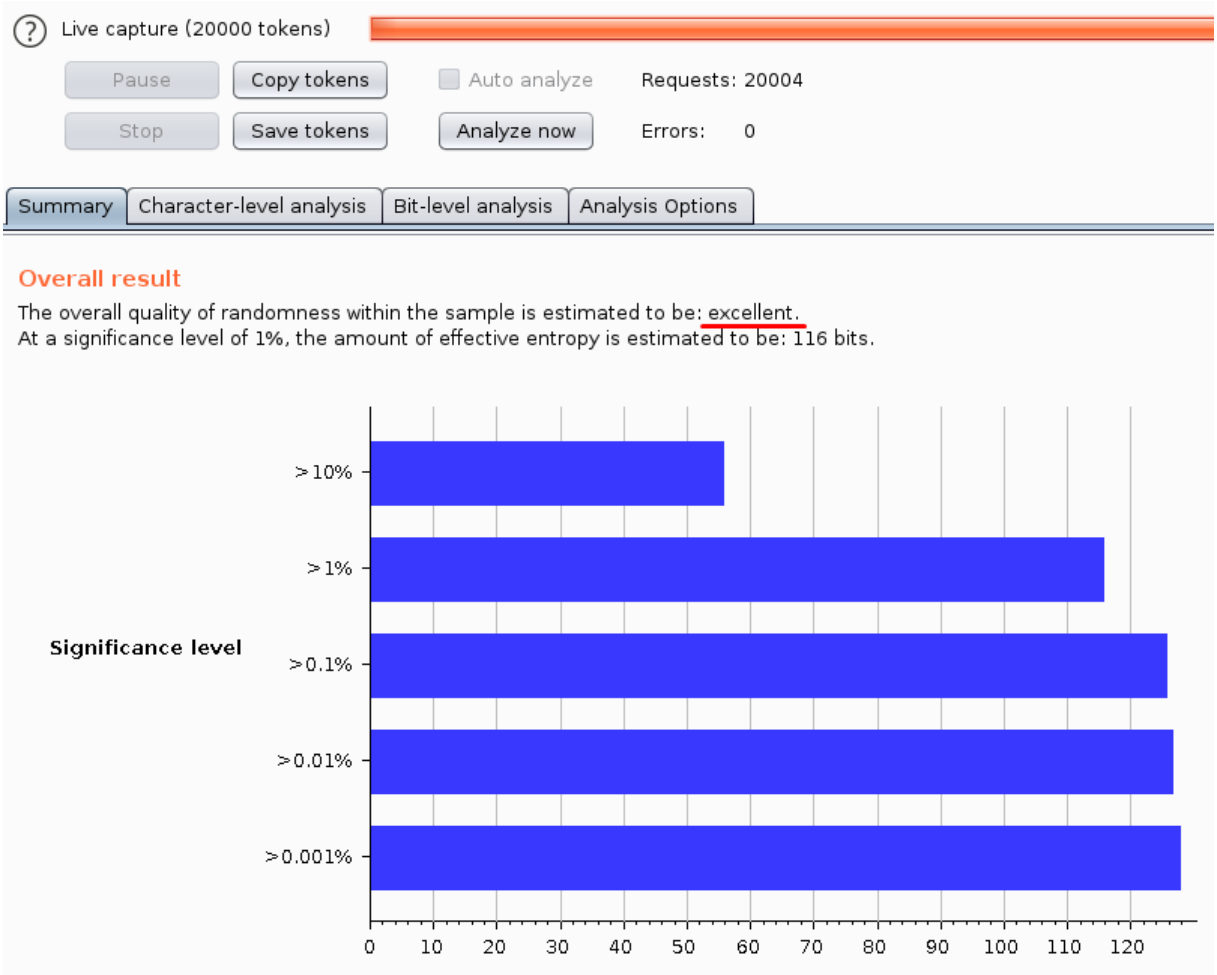
Jednou z možností, jak napadnou relaci, je předpovězení následující *sessionID*. V tomto případě je v DVWA připravena záložka *Weak Session IDs*, kde je tlačítko *Generate* a při každém použití tlačítka se vygeneruje nové *sessionID*. Při zapnuté proxy v Burp Suite je možné sledovat a zobrazovat jednotlivé požadavky, které aplikace odesílá. V případě, že je zabezpečení DVWA nastaveno na *low*, lze *sessionID* lehce předpovídat, protože se s každým požadavkem zvětší o jedna. Ten samý dotaz byl z Burp Suite poslán do *Sequencer,* který je používán k analýze kvality a náhodnosti dat. Po proběhnutí analýzy, bylo výsledkem velmi slabé zabezpečení, což je možné vidět na následujícím obrázku:



Obrázek 10: Analýza předvídatelnosti hodnoty sessionID při nastavení DVWA na low

Zdroj: vlastní

Ten samý ten byl proveden po nastavení zabezpečení DVWA na *high*. Výsledky testu jsou výborné, pokud se ale vygenerovaná *sessionID* podrobí bližšímu průzkumu je zjištěno, že tyto hodnoty odpovídají aktuálnímu datu ve vteřinách. Výsledky analýzy jsou vidět na obrázku níže.



Obrázek 11: Analýza předvídatelnosti hodnoty sessionID při nastavení DVWA na high

Zdroj: vlastní

Pokud útočník bude schopen předpovědět sessionID, může získat přístup do aplikace bez nutnosti přihlášení. S přístupem získá i všechny informace o klientovi z jeho profilu. Tímto krokem se útočník posune o krok blíže a otevřou se mu další možnosti získání dat.

3.2.2 Obrana

Obecná obrana proti útokům tohoto typu je například implementace vícefaktorového ověřování, které zabrání útokům hrubou silou [11]. Tím pádem útočník nezíská přístup do aplikace ani v případě, že se mu podařilo ukradnout cookies platného uživatele. Jedná se tedy o nejefektivnější řešení tohoto problému.

Dalším řešením může být provádění kontrol hesel při jejich vytváření nebo změně. Hesla by měla být otestována na slovníkový útok a zároveň je nutné nastavit dodržování určité délky

hesla. Po malém počtu neúspěšných pokusů o přihlášení je žádoucí nastavit časovou prodlevu pro další pokus o přihlášení a informovat o této skutečnosti administrátora. Tento krok pomůže zabránit útokům hrubou silou.

Co se týká ochrany ID relace, organizace OWASP [19] doporučuje následující:

- Známé a vše říkající názvy jako *PHPSESSID* nebo *CFTOKEN* by se měly nahradit za obecnější název, například *id*. Název totiž může odhalit jaké technologie a programovací jazyky aplikace používá.
- Generování ID relace s vysokou entropií. Mělo by tedy být dostatečně náhodné, aby jej útočník nemohl uhodnout. Za předpokladu, že útočník může vyzkoušet 10 000 různých ID za sekundu a ve webové aplikaci je jich platných celkem 100 000, bude útočníkovi trvat zhruba 290 let, než úspěšně uhodne ID relace s entropií 64 bitů.
- Jak název, tak i obsah ID relace nesmí mít žádný význam, aby útočník nemohl zjistit citlivé informace ať už o webové aplikaci, nebo o uživateli. Doporučuje se ID šifrovat hashovací funkcí jako je například *SHA256*.
- Nastavení atributu *Secure* u cookies souborů, který nařizuje webovým prohlížečům odesílat cookies soubory pouze prostřednictvím šifrovaného připojení HTTPS. Tento mechanismus ochrany relace je povinný, aby se zabránilo odhalení ID relace prostřednictvím útoků typu man-in-the-middle. Účelem je, znemožnit útočníkovi zachytit ID relace z provozu webového prohlížeče.
- Nastavení atributu *HttpOnly* u cookies souborů, který dává webovým prohlížečům pokyn, aby nepovolovaly skripty, což znemožní přístupu ke cookies prostřednictvím *document.cookie*. Tento druh ochrany ID relace je povinný, aby se zabránilo krádeži ID relace pomocí útoků typu XSS. Praktické příklady tohoto druhu útoku jsou popsány v následující kapitole.

3.3 Cross-Site Scripting (XSS)

Tyto útoky jsou typem injekce, při němž jsou škodlivé skripty vkládány na jinak neškodné a důvěryhodné weby [7]. K těmto útokům dochází, když útočník používá danou webovou aplikaci k odesílání svého škodlivého kódu. Obvykle se tak děje ve formě skriptu na straně prohlížeče, kde se nebezpečný kód odešle jinému koncovému uživateli. Prohlížeč uživatele nemá žádný způsob, jak zjistit, že skript není důvěryhodný, a spustí ho. Protože si myslí, že skript pochází z důvěryhodného zdroje, tento skript může přistupovat ke všem cookies souborům,

tokenům relací nebo jiným citlivým informacím, které prohlížeč uchovává a používá na dané webové aplikaci. Tyto skripty mohou dokonce přepsat obsah HTML stránky.

Škodlivý obsah odeslaný do webového prohlížeče je často napsaný v jazyku JavaScript, ale může se jednat také o HTML, Flash nebo jakýkoli jiný typ kódu, který může prohlížeč spustit. Druhy útoků založené na XSS jsou téměř neomezené, ale obvykle zahrnují přenos soukromých dat, jako jsou cookie soubory nebo jiné informace o relaci [20]. Tento útok může mít za následek přesměrování oběti na webový obsah řízený útočníkem nebo provedení jiných škodlivých operací na stroji uživatele. Nedostatky, které umožňují těmto útokům uspět, jsou velmi rozšířené a vyskytují se kdekoli, kde webová aplikace používá vstup od uživatele v rámci výstupu bez ověření.

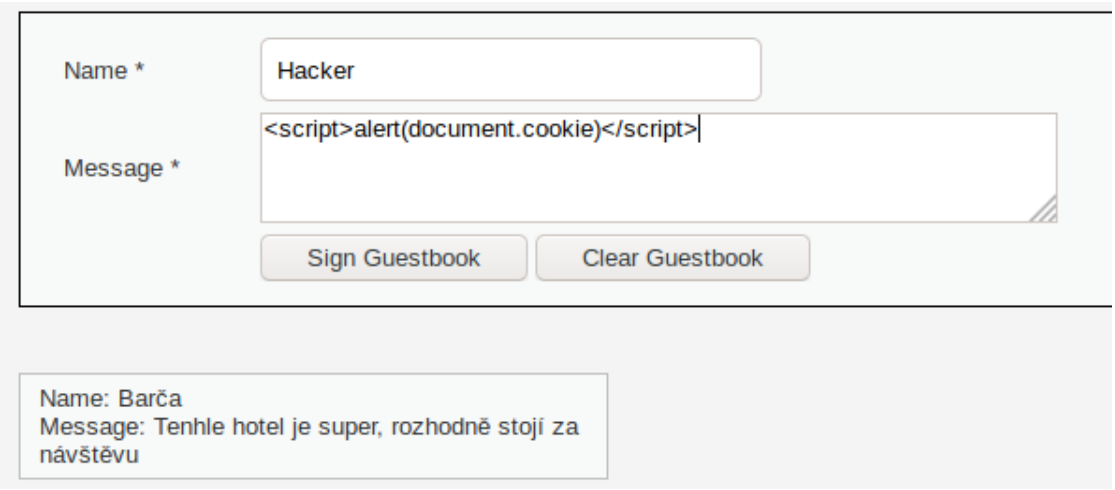
Cross-Site Scripting útoky se dělí do tří základních kategorií podle způsobu, jakým je útok doručen k napadenému uživateli: [20]

1. **Stored XSS** – jedná se o útok, kde je injektovaný skript trvale uložen na cílových serverech, například v databázi, ve fóru zpráv, v protokolu návštěvníků, v poli s poznámkami atd. Oběť poté načte škodlivý skript ze serveru, když požaduje uložené informace.
2. **Reflected XSS** – je útok, kde je injektovaný skript součástí HTTP dotazu na webový server a následně je nasměrován zpět na uživatele jako odpověď serveru. Tyto útoky jsou obětem doručovány například v e-mailové zprávě nebo na jiné webové stránce. Když je uživatel oklamán kliknutím na škodlivý odkaz nebo odesláním speciálně vytvořeného formuláře, útočníkův injektovaný kód přejde na zranitelný web, který se zobrazí uživateli v prohlížeči. Prohlížeč poté škodlivý kód provede, protože se domnívá, že pochází z důvěryhodného serveru.
3. **DOM Based XSS** – tento útok, je proveden v důsledku úpravy prostředí DOM v prohlížeči oběti, což má za následek to, že kód na straně klienta běží neočekávaným způsobem. To tedy znamená, že samotná stránka se nezmění, ale kód na straně klienta obsažený na webové stránce provede dodatečně vloženou funkcionalitu, protože útočník provedl škodlivé úpravy v původní funkci, kterou zkonstruovali vývojáři aplikace.

3.3.1 Praxe DVWA (všechny *XSS* útoky)

1. ***XSS* (*Stored*)**

   Příkladem tohoto útoku mohou být komentáře psané zákazníky hotelu. Na webové stránce je vytvořen formulář, kam zákazník zadá jméno a zprávu. Tento formulář může útočník zneužít k uložení škodlivého kódu na serveru. Do pole pro zprávu stačí doplnit skript *<script>alert(document.cookie)</script>*. Každý uživatel, který si danou stránku zobrazí, bude napaden. V tomto případě se zobrazí informační okno, které obsahuje cookies uživatele, z kterého může útočník získat session ID a vystupovat jako napadený uživatel. Toto informační okno je zobrazeno na obrázku číslo 12.

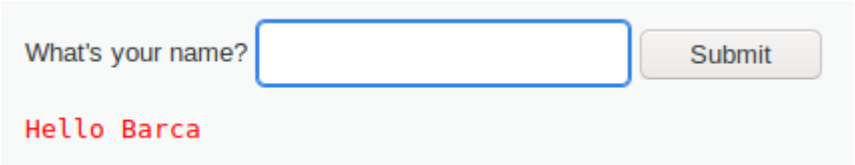


Obrázek 12: Vznik stored XSS útoku

Zdroj: vlastní

2. ***XSS* (*Reflected*)**

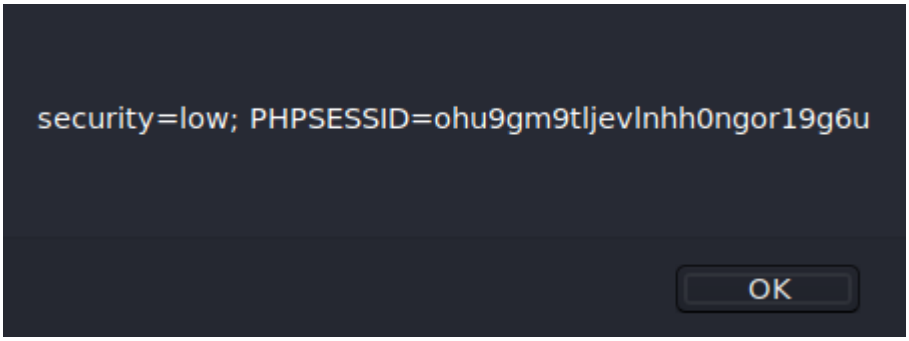
   V tomto případě je v DVWA připraveno pole, do kterého je po zadání a odeslání vypsána hláška na následujícím obrázku.



Obrázek 13: Funkce napadnutelná reflected XSS útokem

Zdroj: vlastní

Do pole je tedy možné vložit HTML tag, který spustí nové informační okno, ve kterém bude vypsán cookies uživatele. Útočník si ve skriptu může vymyslet svou logiku za dosažením získání cookies oběti. Může také vložit obrázek, na který pokud uživatel klikne, útočník opět získá cookies oběti. Skript s informačním oknem: *<script>alert(document.cookie)</script>* a druhý, který vloží obrázek: *<img src=”#” onclick=alert(document.cookie) >*.

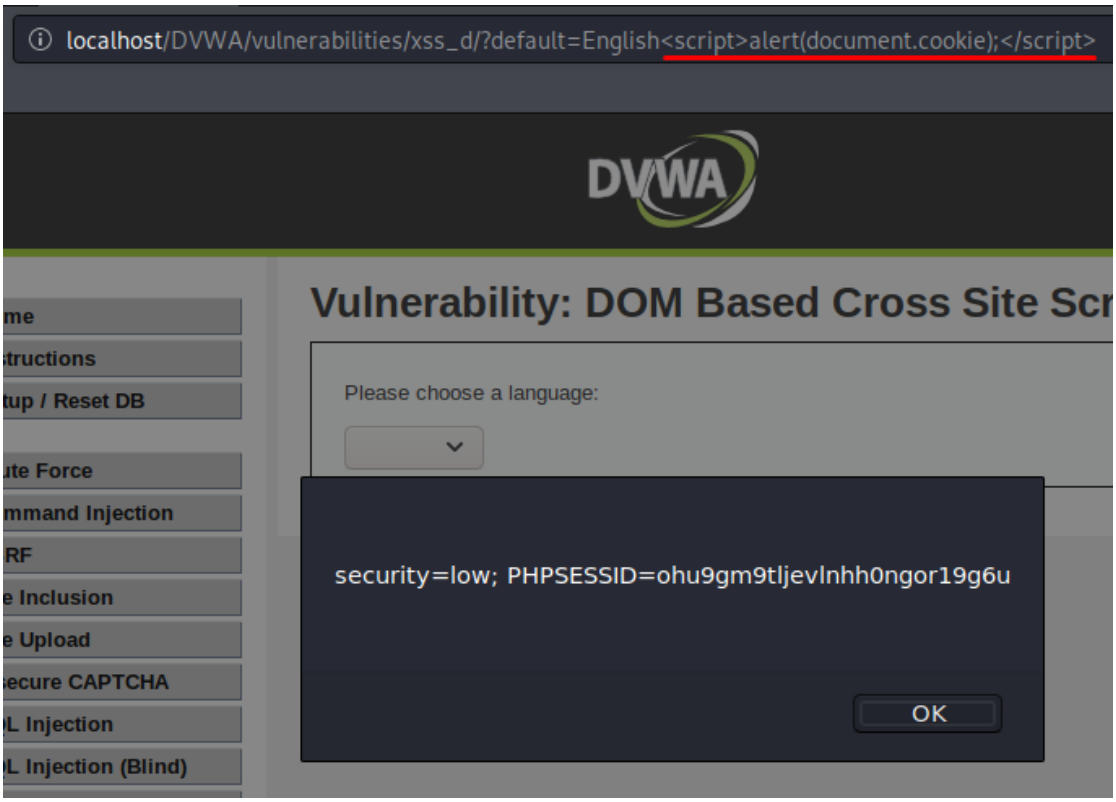


Obrázek 14: Informační okno, které obsahuje cookies oběti

Zdroj: vlastní

3. *XSS (DOM)*

Jako příklad tohoto útoku je rolovací menu, které není v kódu nijak ošetřeno, stačí tedy na konec URL adresy přidat následující skript, který znovu zobrazí v novém okně cookies oběti. Skript je stejný jako v předchozím útoku a na obrázku je označen červeně.



Obrázek 15: DOM Based XSS útok

Zdroj: vlastní

3.3.2 Obrana

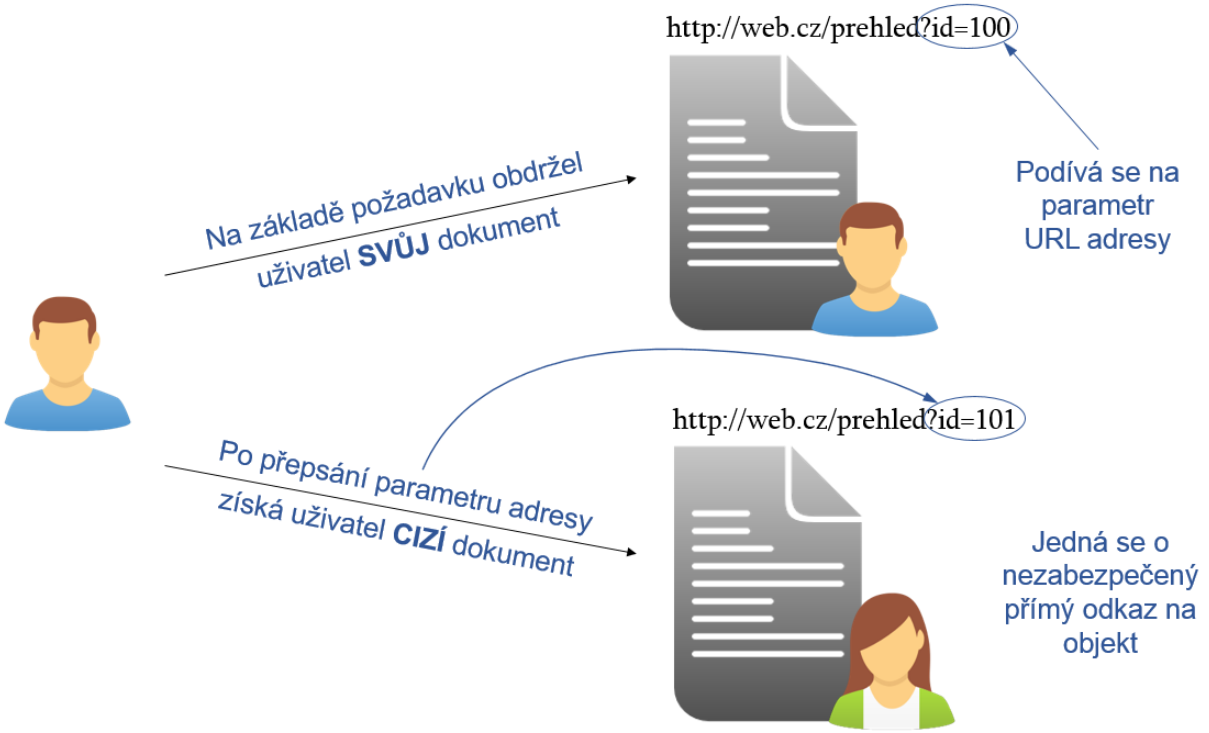
Ochrana před Cross-Site Scripting vyžaduje oddělení nedůvěryhodných dat od uživatelů a funkčního obsahu prohlížeče. Jednou z možností, jak toho lze dosáhnout je používání frameworků, které poskytují ochranu proti XSS. Příkladem může být *Ruby on Rails* nebo *React JS* [10]. Je ovšem potřeba pochopit kde má každý rámec své limity a vhodně je doplnit. Konkrétní seznam s tipy, co dělat nebo naopak nedělat, aby byla obrana efektivní, byla zpracována OWASP ve své *Cheat Sheet*[7] sérii zabývající se XXS.

[7] https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html#xss-prevention-rules

3.4 Nezabezpečený přímý odkaz na objekt

Útočník, který je oprávněným uživatelem systému, jednoduše změní hodnotu parametru, který přímo odkazuje na jeho objekt systému, na cizí objekt, k němuž nemá mít oprávnění. Tato situace nastane, když aplikace používá při vytváření webových stránek skutečný název nebo klíč nějakého objektu [10]. Aplikace ne vždy ověřují, zda je uživatel oprávněn přistupovat k cílovému objektu, což má za následek nezabezpečený přímý přístup k objektu.

Chyby tohoto typu je snadné odhalit, stačí manipulovat s hodnotami parametrů v URL adrese a skutečnost, zda jsou autorizace řádně kontrolovány, se rychle projeví. Tyto chyby mohou ohrozit veškerá data, na která parametr odkazuje. Pokud je odkaz na objekt předvídatelný, útočník snadno získá přístup ke všem datům tohoto typu.



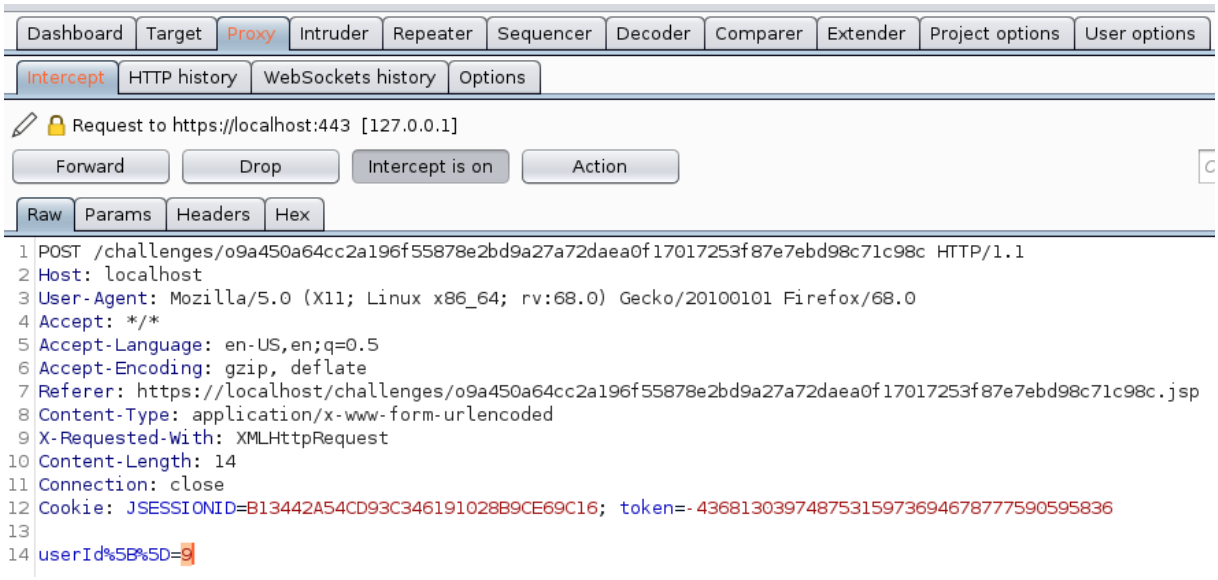
Obrázek 16: Příklad nezabezpečeného přímého odkazu na objekt

Zdroj: vlastní

3.4.1 Praxe Security Shepherd (*Insecure Direct Object Reference Challenge 1*)

Úkolem této lekce bylo najít skrytou zprávu uživatele, který není zobrazený v menu. K tomuto účelu byl použit Burp Suite pro zachycení komunikace. U každého uživatele byla odchycena komunikace, odesílající informace o profilu uživatele. Každý uživatel měl jinou hodnotu

ID, po odchycení profilů všech uživatelů bylo zjištěno, že jejich ID jsou: 1, 3, 5, 7, 9, bylo tedy snadné uhodnout, že další ID uživatele bude 11. Při odchycení jakéhokoli profilu tedy byla hodnota ID přepsána na 11 a poslána dál. Výsledkem je úspěch v podobě klíče. Útočník může tímto způsobem získat přístup k citlivým datům a údajům zákazníka.



Obrázek 17: Požadavek v Burp Suite, kde je vidět ID uživatele

Zdroj: vlastní

3.4.2 Obrana

Ochrana každého objektu, který je dostupný uživatelům, jako je například číslo objektu nebo název souboru, vyžaduje vhodný výběr metody ochrany. Pokud je použit přímý odkaz na objekt z nedůvěryhodného zdroje, musí obsahovat kontrolu přístupu a musí zaručit, že uživatel má k požadovanému objektu práva. Toto téma úzce souvisí s obranou popsanou v kapitole o chybách v řízení úrovní přístupů.

Je žádoucí používat nepřímé odkazy na objekty pro každého uživatele nebo relaci zvlášť. Zabrání to přístupu útočníkům, kteří se zaměřují na tento druh útoků. Příkladem může být použití rozbalovacího seznamu s daným počtem schválených hodnot pro daného uživatele, místo vkládání databázového klíče zdroje. K označení klíče, který si uživatel vybral, lze použít čísla od jedničky až po nejvyšší schválenou hodnotu. Aplikace poté musí namapovat nepřímý odkaz, který je specifický pro daného uživatele zpět na skutečný databázový klíč na serveru. K tomuto

účelu je vhodné použít OWASP Enterprise Security API (ESAPI), který umožnuje programátorům použít různé druhy mapování na místo přímých odkazů na objekt webové aplikace. Více informací o ESAPI[8]. [10]

3.5 Nezabezpečená konfigurace

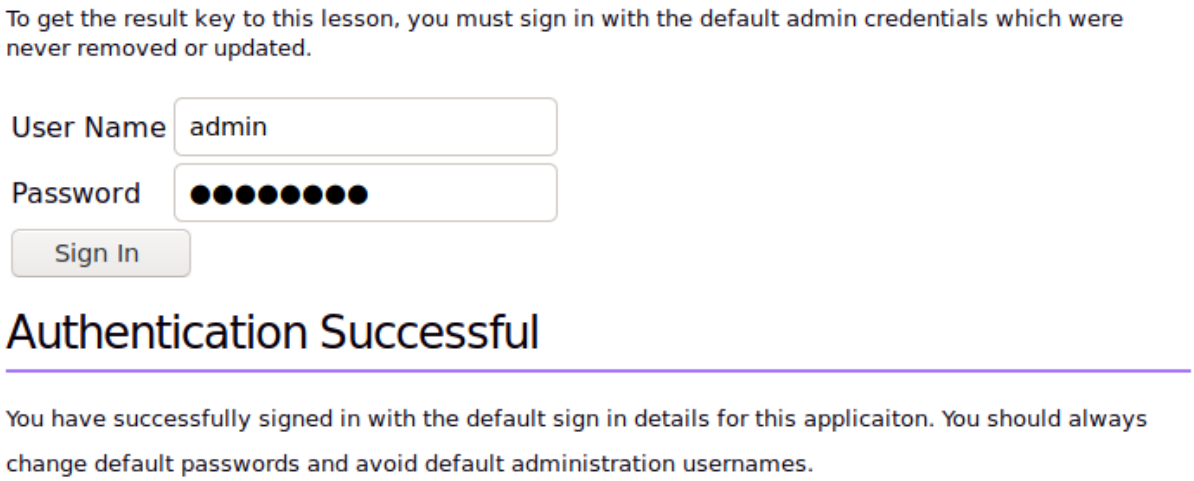
Chybně zabezpečená konfigurace vzniká, když jsou nastavení zabezpečení definována, implementována a udržována jako výchozí [10]. Toto může nastat, když je nastavení z vývoje předáno do produkce. Dobré zabezpečení vyžaduje bezpečnou konfiguraci definovanou a nasazenou pro aplikaci, webový server, databázový server a platformu. Stejně důležité je mít aktuální software, který může záplatovat některé chyby z předchozí verze.

Útočník může záměrně vyhledávat výchozí účty, nepoužívané stránky, neopravené chyby, nechráněné soubory nebo adresáře, aby získal neoprávněný přístup nebo informace o systému [21]. Špatně zabezpečená konfigurace se může objevit na jakékoliv úrovni aplikace včetně platformy, webového serveru, aplikačního serveru, databáze, frameworku nebo vlastního kódu. Zranitelnosti tohoto typu často poskytují útočníkovi neautorizovaný přístup k některým datům nebo funkcím systému a mohou vést i k úplné kontrole nad systémem. K detekci chybějících záplat, chybné konfigurace, výchozích účtů a nepotřebných služeb je možné využít automatizovaných skenerů.

3.5.1 Praxe Security Shepherd (*Security Misconfiguration Lesson*)

V tomto případě, je otestováno, jak jednoduché je uhodnout výchozí přihlašovací údaje, které mohou vést například k přístupu do aplikace. Je zde vytvořen přihlašovací formulář, kde stačí tyto údaje správně uhodnout a úspěšně se přihlásit. Aplikace sama sděluje, zda je špatně heslo, nebo uživatelské jméno, takže po vyzkoušení jména *root*, které nebylo správně, byla třeba do černého volba jména *admin*. Vyzkoušením malého počtu jednoduchých a základních hesel bylo dospěno k závěru, že správné heslo je *password*.

[8] https://owasp.org/www-project-enterprise-security-api



Obrázek 18: Úspěšné přihlášení pod výchozími přihlašovacími údaji

Zdroj: vlastní

3.5.2 Obrana

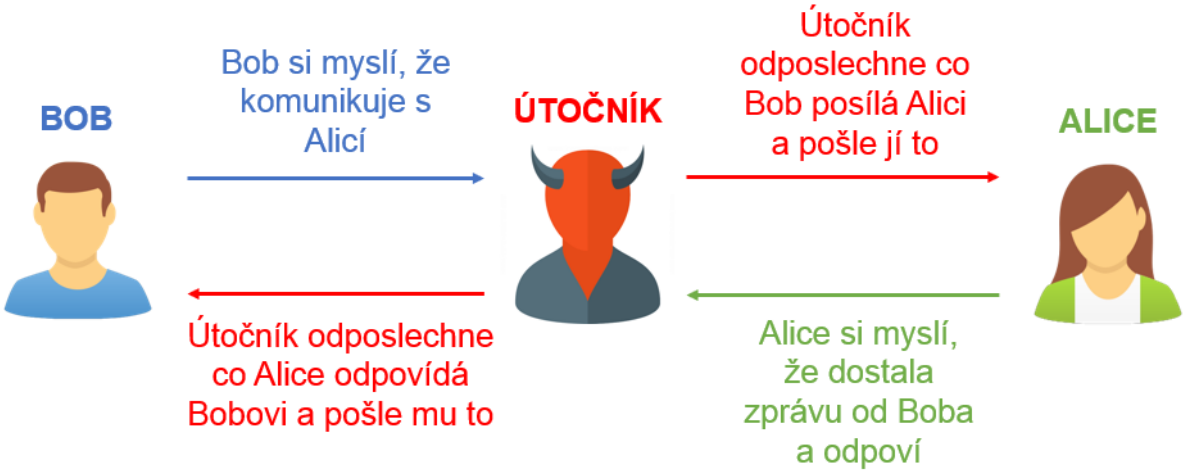
Ubránit se tomuto útoku je velmi jednoduché, stačí dodržovat obecné zásady pro tvorbu hesel. Heslo by mělo být dlouhé alespoň 10 znaků, obsahovat kombinaci písmen, čísel a speciálních znaků, nesmí být odvoditelné od uživatele a nemělo by obsahovat reálné slovo. Za tímto účelem je nejlepší si nějaké slovo splňující požadavky vymyslet, nebo použít sofistikované stránky[9], které heslo vymyslí za uživatele a navrhnou pomůcku pro zapamatování hesla.

Obecná obrana proti tomuto druhu útoku je alespoň částečná automatizace kontroly aplikace před jejím nasazením do produkce. Tato kontrola by měla zahrnovat i různé záplaty a aktualizace aplikace. Pokud je potřeba konfigurace většího množství serverů, může tento úkol zastat automatizace konfigurace serverů, případně lze použít nástroj pro replikaci disků. Důležité je také sledování používaných komponent a jejich verzí a aktualizací. Účelem je monitorování možných záplat v použitých technologiích a jejich následná aktualizace na verzi, která chybu opraví. Vhodné je také aktuální sledování publikovaných bezpečnostních chyb a jejich ověření v dané aplikaci. Jak již bylo zmíněno, je nutné mít na paměti, že kontrola musí probíhat na všech úrovních aplikace. [10]

[9] https://passwordsgenerator.net

3.6 Expozice citlivých dat

Útočníkům se obvykle nepodaří přímo prolomit šifrovaná data, ale například kradou klíče, používají útoky typu man-in-the-middle, nebo kradou nešifrovaná data ze serveru, ať už během přenosu, nebo z uživatelova prohlížeče. Podstatou útoku MITM je útočníkova snaha odposlouchávat komunikaci mezi účastníky tak, aby se stal aktivním prostředníkem. Většinou k tomuto účelu využívá podvržený certifikát. Pro bližší představu vizte následující obrázek:



Obrázek 19: Princip útoku man-in-the-middle

Zdroj: vlastní

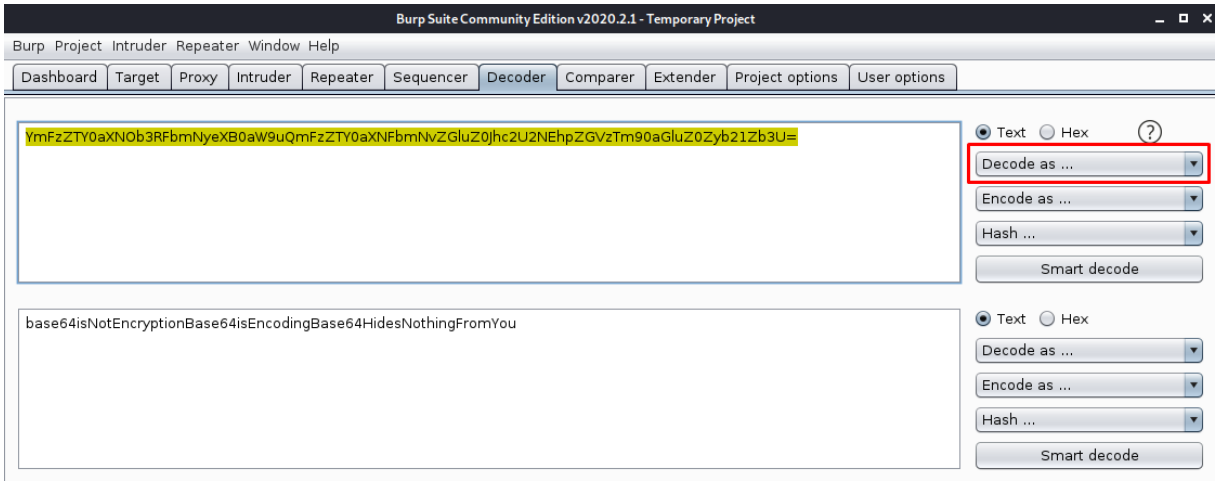
Nejčastější chybou je, že citlivá data nejsou šifrovaná. Časté je generování a správa slabých šifrovacích klíčů a používání slabých algoritmů, zvláště pak slabých technik hashování hesla [17]. Uložení nějakých citlivých dat v databázi bez jejich hashování by se nemělo nikdy stát. Pokud se například do databáze neuloží pouze hash hesla, tak v případě, že do ní útočník získá přístup, bude mít přístup ke všem heslům. Chyba často ohrožuje veškeré údaje, které by měly být chráněné. Obvykle se jedná o informace, které obsahují citlivé údaje, jako jsou zdravotní záznamy, pověření, osobní údaje nebo kreditní karty.

Příkladem může být i rozdíl mezi HTTP a HTTPS. Komunikační protokol HTTP slouží k přenosu dat na internetu. Tato verze není nijak chráněná, kdokoli tedy může metodou man-in-the-middle odposlouchávat komunikaci a bez problémů ji číst, protože není nijak chráněna

šifrováním. Z tohoto důvodu vznikl protokol HTTPS. Zde je přenos dat zajištěn SSL certifikátem, s jehož pomocí se šifrují data přenášená přes protokol HTTPS. Komunikaci tedy přečte pouze odesílatel a příjemce, kteří mají správný klíč pro provedení dešifrování. [22]

3.6.1 Praxe Security Shepherd (*Insecure Cryptographic Storage Lesson*)

Cílem této lekce je ukázat, že base64 není druh šifrování, ale kódování a zabezpečení dat tímto způsobem je téměř stejné, jako kdyby data nebyla vůbec zabezpečena. Klíč k dokončení lekce je tedy poskytnut zakódovaný s pomocí base64. Zjištění textu se dá provést jednoduše zkopírováním poskytnutého zakódovaného řetězce do záložky *Decoder* v aplikaci *Burp Suite*, kde se vybere *Decode as base64*. Jak je vidět na obrázku níže, výsledkem je prostý text, který slouží jako klíč k dokončení lekce. Ponaučení z této lekce je, že všechna citlivá data by měla být šifrována tak, aby je útočník nemohl prolomit. To znamená, používat jen opravu účinné druhy šifrování. Nutno podotknout, že do této kategorie se rozhodně nepatří MD5, jehož prolomení bylo uvedeno v kapitole o SQL injektování.



Obrázek 20: Dekódování base64 v aplikaci Burp Suite

Zdroj: vlastní

3.6.2 Obrana

Zde je nejdůležitějším krokem stanovení citlivých dat, které je potřeba ochránit. U takových dat je vhodné zakázání ukládání do mezipaměti. Data, která nejsou potřebná, je lepší neukládat vůbec, aby nemohlo dojít k jejich zneužití. Citlivá data, která je nutné poslat po internetu, se musí zašifrovat například pomocí kryptografických protokolů, jako je TLS nebo SSL. [23]

Co se týče ochrany hesel, doporučuje se použití hashovací funkce jako je například *Bcrypt* nebo *Argon2d*. Tyto algoritmy automaticky používají náhodně generované řetězce, které přidají k heslu těsně před jeho zahashováním [24]. Tato část se nazývá přidání soli a je pro každého uživatele jedinečná. Útočník tedy musí prolomit hash jeden po druhém pomocí příslušné soli, oproti vypočítání hashe jednou a porovnáním ho s každým uloženým hashem. Dalším benefitem je, že přidání soli znemožňuje zjištění, zda mají dva uživatelé stejné heslo.

3.7 Chyby v řízení úrovní přístupů

V rámci základních bezpečnostních mechanismů aplikace je řízení přístupu postaveno na autentizaci a správě relací. Aplikace nejprve ověří totožnost uživatele a pak si to určí posloupnost žádostí, které od tohoto uživatele obdržela. Je tedy nutné implementovat řízení přístupu pro povolení, nebo zamítnutí akce. Cílem je zjistit, zda v dané situaci s danými právy uživatele je možné danou akci provést. Žádoucí je, nepředběhnout kroky tak, aby například nedošlo k navýšení práv uživatele dříve, než je třeba, což může mít za následek provedení akce, ke které by nemělo dojít [10]. Řízení přístupu je kritickým obranným mechanismem aplikace, protože je odpovědné za tato klíčová rozhodnutí. Když jsou chybně implementované, může útočník ohrozit celou aplikaci a převzít kontrolu nad její funkčností a získat přístup k citlivým datům všech ostatních uživatelů. Testování řízení úrovní přístupu nelze provádět automatizovaně, vždy je potřeba, aby testy prováděl člověk.

Příkladem napadení může být útočník, který využije nezabezpečeného přímého odkazu na objekt popsaného výše. Díky úpravě parametru URL adresy může útočník získat přístup k dokumentům, ke kterým by neměl mít přístup. Hlavním cílem tohoto útoku jsou administrátorské funkce, ke kterým se útočník snaží dostat.

Řízení přístupu lze rozdělit do tří hlavních kategorií, které definují, jak s přístupem nakládat:

1. **Vertikální** – umožňuje různým uživatelům přístup k různým částem funkčnosti aplikace. V nejjednodušším případě to obvykle zahrnuje rozdělení na běžné uživatele a administrátory. Ve složitějších případech mohou vertikální přístupové kontroly zahrnovat rozmanité uživatelské role udělující přístup ke specifické funkci, kde je každý uživatel přidělen k jedné roli nebo kombinaci různých rolí. Příklad:
    - Obyčejný uživatel, který může vykonávat administrativní funkce.
    - Obyčejný úředník, který může platit faktury jakékoli velikosti.
2. **Horizontální** – uživatelé mají přístup k určité části z většího rozsahu zdrojů stejného typu. Například aplikace emailového klienta může umožnit adresátovi číst jeho

e-mail, ale nikdo jiný ho číst nemůže. Nebo online bankovnictví dovolí klientovi převádět peníze pouze z jeho účtu, ne z cizího. Konkrétnější příklad:

- Uživatel, který může číst e-maily nejen své, ale i jiných lidí.
- Platební úředník, který může zpracovávat faktury jiné organizace než vlastní.

3. **Závislé na kontextu** – zajišťuje, že přístup uživatelů je omezen jen na to, co je povoleno vzhledem k aktuálnímu stavu webové aplikace. Například, pokud uživatel provádí více fází procesu najednou, může mu být zabráněno provádět akce, které jsou závislé na některých jiných.

3.7.1 Praxe Security Shepherd (*Failure To Restrict URL Access Challenge 1*)

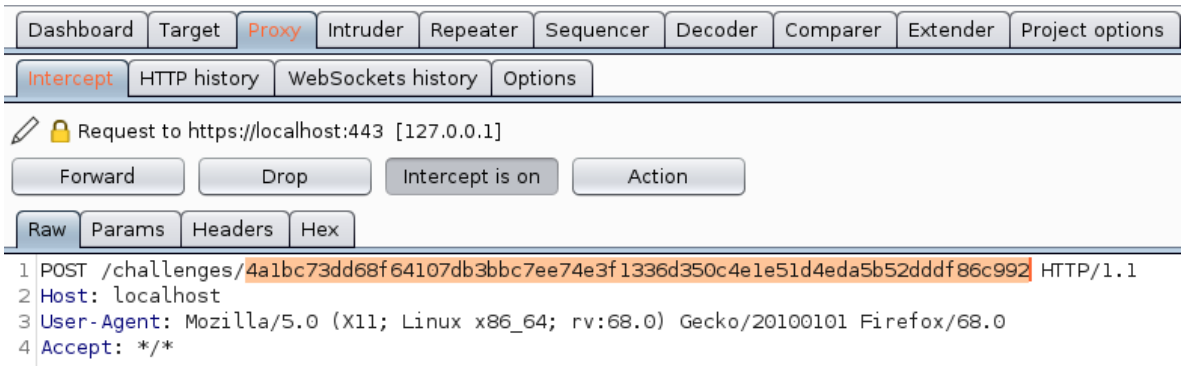
Formulář této stránky je jednoduchý, obsahuje pouze tlačítko s nápisem *Get Server Status*. Cílem je, získat informaci o serveru z pohledu administrátora. Pokud na tlačítko klikne obyčejný uživatel, zobrazí se informace, že server je v pořádku a není zde nic k vidění. Při bližším prozkoumání zdrojového kódu dané stránky aplikace je možné si povšimnout dvou rozdílných formulářů. Jedná se o formuláře s názvy *leForm* a *leAdminForm*, jak sám název napovídá, jeden formulář je přístupný pouze administrátorovi. Jak je vidět na následujícím obrázku, oba formuláře obsahují *url* a po porovnání, je zjištěno, že se liší. Byla nalezena chyba, využita bude přes aplikaci Burp Suite, v němž bude po kliknutí na tlačítko odchycen odeslaný požadavek. V Burp Suite je tento požadavek upraven na administrátorovu hodnotu nalezenou v kódu stránky. Následně je požadavek odeslán a výsledkem je úspěšné dokončení lekce.

```
$("#leForm").submit(function() {
    $("#submitButton").hide("fast");
    $("#loadingSign").show("slow");
    $("#resultsDiv").hide("slow", function() {
        var ajaxCall = $.ajax({
            type: "POST",
            url: "4a1bc73dd68f64107db3bbc7ee74e3f1336d350c4e1e51d4eda5b52dddf86c99",
            data: {userData: "4816283",},
               async: false
        });
        if (ajaxCall.status == 200) {
            $("#resultsDiv").html(ajaxCall.responseText);
        } else {
            $("#resultsDiv").html("<p> An Error Occurred: " + ajaxCall.status + " " +
ajaxCall.statusText + "</p>");
        }
        $("#resultsDiv").show("slow", function() {
            $("#loadingSign").hide("fast", function() {
                $("#submitButton").show("slow");
            });
        });
    });
   });

$("#leAdminForm").submit(function() {
    $("#submitButton").hide("fast");
    $("#loadingSign").show("slow");
    $("#resultsDiv").hide("slow", function() {
        var ajaxCall = $.ajax({
            type: "POST",
            url: "a1bc73dd68f64107db3bbc7ee74e3f1336d350c4e1e51d4eda5b52dddf86c992",
            data: {userData: "4816283",},
            async: false
        });
        if (ajaxCall.status == 200) {
            $("#resultsDiv").html(ajaxCall.responseText);
        } else {
         $("#resultsDiv").html("<p> An Error Occurred: " + ajaxCall.status + " " +
ajaxCall.statusText + "</p>");
        }
        $("#resultsDiv").show("slow", function() {
            $("#loadingSign").hide("fast", function() {
                $("#submitButton").show("slow");
            });
        });
    });
});
```

Obrázek 21: Zranitelný kód umožňující přístup k administrátorským funkcím

Zdroj: vlastní



Obrázek 22: Přepsání URL adresy odchyceného požadavku v aplikaci Burp Suite

Zdroj: vlastní

Útočník může podobným způsobem získat přístup k účtu administrátora a mít tak kontrolu nad celou aplikací. Tato kapitola úzce souvisí s kapitolou o nezabezpečeném přímém odkazu na objekt. Obě kapitoly se zabývají chybnou implementací práv uživatele. Tento druh útoků nelze plně zautomatizovat, protože pouze člověk pozná, zda získal přístup k informacím, ke kterým by přístup mít za normálních okolností neměl.

3.7.2 Obrana

Zde je hlavním úkolem stanovení a správa oprávnění uživatelů. Autorizace by měla být nastavena tak, aby implicitně odmítla každou funkcionalitu a povolila ji jen daným uživatelským rolím [10]. Dobrým řešením je provést autorizaci na straně klienta i na straně serveru v kódu aplikace. To zabrání útočníkovi, aby si mohl změnit uživatelskou roli. Jak je vidět na praktické části této podkapitoly, nevhodným řešením je vpisovat oprávnění přímo do kódu. Také nelze předpokládat, že úpravy pouze ve vizuální části aplikace budou plnit funkci ochrany před tímto útokem.

3.8 Cross-Site Request Forgery (CSRF)

CSRF je útok, který nutí koncového uživatele k provedení nechtěných akcí ve webové aplikaci, ve které jsou tyto akce autorizovány [10]. Útočník tedy sebere identitu a privilegia oběti, aby mohla vykonávat nežádoucí funkci jménem oběti. Na většině webů požadavky prohlížeče automaticky zahrnují veškerá pověření spojená s webem, jako jsou cookies uživatele relace, IP adresa, pověření domény Windows atd. Pokud je tedy uživatel aktuálně ověřen na webové

stránce, nebude mít server žádný způsob, jak rozlišit padělaný požadavek zaslaný obětí a legitimním požadavkem pod stejnou identitou.

Cross-Site Request Forgery útočí na cílové funkce, které způsobují změnu stavu na serveru, jako je změna e-mailové adresy nebo hesla oběti nebo nákup něčeho. Přinutit oběť k načtení dat neprospívá útočníkovi, protože útočník nedostane žádnou odpověď, oběť ano. Útoky typu CSRF jako takové cílí na požadavky na změnu stavu.

Někdy je možné uložit útok CSRF na samotnou zranitelnou stránku. Tyto chyby zabezpečení se nazývají „uložené chyby CSRF“. Toho lze dosáhnout jednoduchým uložením značky *img* nebo *iframe* do pole, které přijímá HTML. Pokud tato útočníkova akce umožní uložení CSRF útoku na webu, zvyšuje se závažnost útoku. Pravděpodobnost je zvýšena, protože oběť má větší šanci zobrazit stránku obsahující útok než některou náhodnou stránka na internetu. Pravděpodobnost se také zvyšuje, protože oběť je již na webu ověřena. [25]

Pokud útočník v kombinaci s CSRF využije sociálního inženýrství (například odesláním odkazu prostřednictvím e-mailu nebo chatu) může útočník přimět uživatele webové aplikace k provedení akcí podle vlastního výběru. Pokud se stane obětí normální uživatel, může ho úspěšný útok nutit, aby provedl požadavek na změnu stavu, jako je převod financí, nebo změna e-mailové adresy. Pokud se administrativní účet stane obětí, může útok ohrozit celou webovou aplikaci. Příkladem může být vložení funkčního HTML tagu do obrázku, který následně převede finance oběti na účet útočníka.

3.8.1 Praxe DVWA (*CSRF*)

Na ukázání základní funkce útoku Cross-Site Request Forgery má DVWA připraven formulář na změnu hesla. Je možné si přes něj obyčejně změnit heslo pro přihlášení do aplikace. Pokud se však trochu poupraví, je možné přepsat heslo uživatele na zcela jiné.

Je potřeba zobrazit si zdrojový kód stránky a najít část, která se týká dané funkcionality. Podle této části se vytvoří nový formulář, obsahující nadpis *Klikni zde*, pod kterým najdeme tlačítko. Pokud uživatel na toto tlačítko klikne, nové heslo se automaticky změní na hodnotu *napadeno*, jak je uvedeno v nové části kódu. Když se chce uživatel znovu přihlásit, jeho staré heslo nefunguje a on netuší, že jeho heslo bylo změněno na nové, konkrétně na *napadeno*.

V tomto případě hraje uživatel DVWA role jak oběti, tak útočníka. Útočník si pozměnil kód, uložil ho jako soubor html v počítači. Oběť otevřela soubor v prohlížeči a klikla na tlačítko ve formuláři, což mělo za následek přesměrování zpět na stránku, která informovala o změně

hesla. Oběť však nikde nové heslo nezadala, to udělal útočník za ni a díky tomu teď má přístup do aplikace. Původní kód a jeho změnu je možné vidět níže.

```
<form action="#" method="GET">
   New password:<br />
   <input type="password" AUTOCOMPLETE="off" name="password_new"><br />
   Confirm new password:<br />
   <input type="password" AUTOCOMPLETE="off" name="password_conf"><br />
   <br />
   <input type="submit" value="Change" name="Change">
</form>
```

Obrázek 23: Kód aplikace, který změní heslo uživatele na jím vybranou hodnotu

Zdroj: vlastní

```
<form action="http://localhost/DVWA-master/vulnerabilities/csrf/? "method="GET">
   <h1>Klikni zde</h1>
   <input type="hidden" AUTOCOMPLETE="off" name="password_new" value="napadeno">
   <input type="hidden" AUTOCOMPLETE="off" name="password_conf" value="napadeno">
   <input type="submit" value="Change" name="Change">
</form>
```

Obrázek 24: Kód útočníka, který změní heslo oběti na hodnotu *napadeno*

Zdroj: vlastní

V reálné situaci však k útoku dojde pravděpodobně jinak. V podobném scénáři oběť obdrží například email od útočníka, který bude obsahovat odkaz na změnu hesla v dané aplikaci. Když na něj oběť klikne, je přesměrována na stránku útočníka, která vypadá stejně jako daná aplikace. Oběť změní přihlašovací údaje na podvrhnuté stránce, které se změní na hodnotu, kterou jim útočník zadá, a celé to se odešle oficiální stránce pro ověření. Tento scénář se například může změnit tak, že aplikace donutí oběť k převodu peněz ve prospěch útočníka.

3.8.2 Obrana

V tomto případě pomůže pouze obezřetnost uživatele, musí si být vědom hrozícího nebezpečí a nepřistupovat ke každému odkazu který mu byl zaslán. Zpozornit by měl uživatel především pokud se nevyskytuje v kruzích, kde se hovoří cizím jazykem a email je napsaný například v Anglickém jazyce, zvýšená opatrnost je na místě i v případě že je email napsán lámaným českým jazykem. Tyto druhy emailů mohou přijít jak z naprosto cizího účtu, tak jako je uživatel může obdržet od někoho koho zná a jehož byl kompromitován.

Doporučuje se také použití CSRF tokenů u všech požadavků na změnu stavu (požadavky, které způsobují akce na webu) a jejich následné ověření na backendu. Tyto obranné tokeny byly zabudovány do mnoha frameworků. Je vhodné prozkoumat, zda framework, který je v aplikaci použit, má možnost ochrany CSRF ve výchozím nastavení, místo pokusu o vytvoření vlastního systému generování tokenů. Například .NET má vestavěnou ochranu, která přidává token do zdrojů zranitelných na CSRF útok. Je nutné mít na paměti, že před použitím této vestavěné funkce, která generuje CSRF tokeny, je potřeba správně nakonfigurovat správu klíčů a tokenů. Pokud aplikace nepoužívá žadný framework s touto orchanou, je možné přidat externí komponenty, které ji implementují. Příkladem pro *Javu* může být *Spring Security* a pro *PHP* a *Apache* je to například *CSRFProtector Project.* [26]

Pro cookie soubory relací je vhodné použít atribut *Cookies SameSite*, který zabrání tomu, aby prohlížeč odeslal cookie soubor na cílový web. Například pro web podobný jako je *GitHub* by to znamenalo, že pokud přihlášený uživatel otevře odkaz na soukromý *GitHub* projekt zveřejněný kolegou na firemním e-mailu, *GitHub* neobdrží cookie soubor dané relace a uživatel nebude mít přístup k projektu.

3.9 Použití známých zranitelností komponent

Nabídky komponent pro webové aplikace jsou stále komplexnější a některé zranitelné komponenty, jako například knihovny, mohou být identifikovány a zneužity automatickými nástroji. Skupina útočníků se tím rozšiřuje z těch, co jdou cestou cílených útoků také o aktéry, kteří jednají náhodně. Útočník najde buď automatickým skenováním, nebo ručním procházení stránek nějakou slabou komponentu. Útok poté přizpůsobí nalezené zranitelnosti a provede ho. Útok je obtížnější, pokud je zranitelná komponenta hlouběji v aplikaci. Těmito nedostatky disponuje velké množství aplikací, protože vývojáři nevěnují dostatečnou pozornost tomu, zda jsou jejich komponenty a knihovny aktuální.

V mnoha případech vývojáři ani nevědí, jaké komponenty využívají, tím pádem neznají ani jejich verze a situaci ještě zhoršují závislosti mezi jednotlivými komponentami. V aplikaci se tedy mohou vyskytnout jakékoli zranitelnosti, například injektování, prolomení přístupu do aplikace, XSS a podobné. Nebezpečí poté spočívá v částečném, nebo úplném převzetím kontroly nad webovou stránkou používající danou komponentu a kompromitaci jejích dat.

Teoreticky by mělo být jednoduché zjistit, jestli je nějaká zranitelná komponenta v dané aplikaci použita. Bohužel informace o zranitelnostech v komerčním, nebo open source softwaru ne vždy jasně specifikují, jaké verze komponent jsou zranitelné. Navíc ne ve všech knihovnách

se využívá srozumitelný způsob číslování verzí. Největší problém ovšem představuje to, že ne všechny zranitelnosti jsou nahlášeny do centrálního systému, který lze snadno prohledat. Je ovšem potřeba zmínit, že na některých[10] stránkách se postupem času vyhledává stále snáze. Zda systém obsahuje nějakou zranitelnou komponentu, je možné zjistit procházením těchto databází a neustálým sledováním e-mailových oznámení o možných zranitelnostech. Pokud nějaká komponenta zranitelnost obsahuje, je nutné zkontrolovat, jestli daný kód příslušnou část této komponenty využívá a jaký dopad by mohlo mít její zneužití. [10; 21]

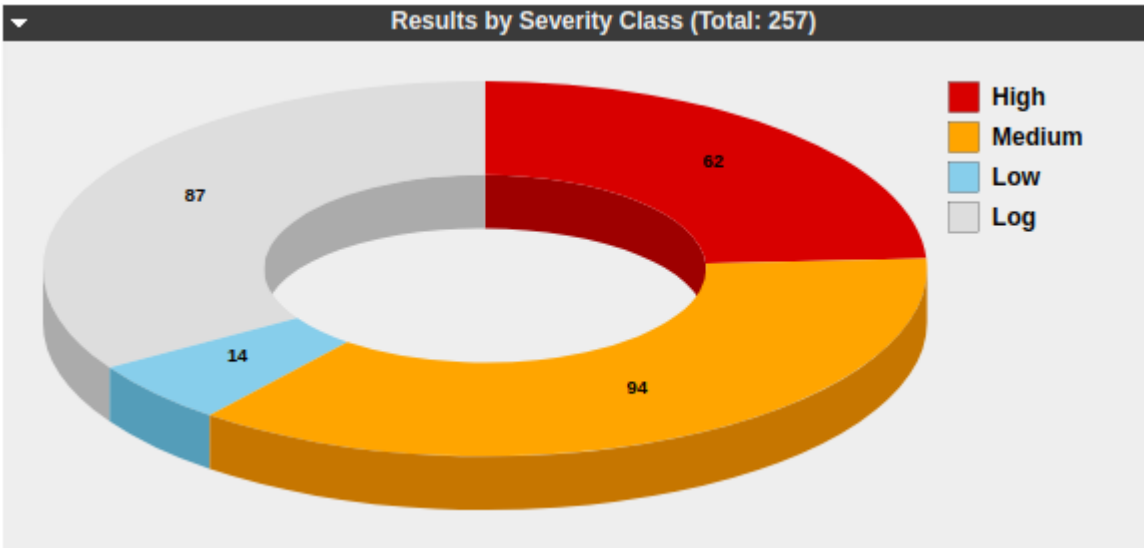
3.9.1 Praxe OpenVAS (DVWA na Ubuntu serveru)

K odhalení zranitelností komponent bylo využito aplikace OpenVAS, která provedla automatické skenování webové aplikace DVWA, která byla spuštěna na Ubuntu. Pro provedení skenu bylo nutné vložit SSH klíč z testovacího stroje do autorizovaných klíčů na serveru. Po úspěšném spuštění a načtení uživatelského rozhraní na daném portu localhostu je potřeba nastavit nový sken na server, na kterém je spuštěna daná webová aplikace.

V uživatelském rozhraní OpenVAS pod záložce *Configuration* → *Credentials* → a pod symbolem hvězdičky vytvořit nový. Položku *Type* je nutné nastavit na *Username + SSH key*, vyplnit položku *Username* a do *Private Key* vložit soubor obsahující privátní klíč stroje, na kterém je OpenVAS spuštěn. Následně je potřeba nastavit cíl, který má OpenVAS testovat, je potřeba ho tedy vytvořit v *Configuration* → *Targets* → *symbol hvězdičky*. V položce *Hosts* zaškrtnout *Manual* a zadat IP adresu serveru a položku *SSH* nastavit na název již vytvořeného *Credentials*. Všechny ostatní nastavení zanechat nedotčené. Nakonec je potřeba vytvořit nový sken pod záložkou *Scans* → *Tasks* → *symbol hvězdičky.* Položku *Scan Targets* je nutné nastavit na *Target* vytvořený v předchozím kroku a položka *Scan Config* byla nastavena na *Full and very deep* a ostatní položky zůstaly jako výchozí. Vše bylo správně nastaveno a nový sken stačí pouze spustit a čekat na výsledky.

Po dokončení skenování bylo zjištěno 62 závažných zranitelností. Ty byly testovány na základě přítomnosti komponent na testovaném stroji. Tato informace je v následujícím výsledném obrázku definována jako *Log*.

[10] http://cve.mitre.org/
https://nvd.nist.gov/home



Obrázek 25: Výsledky skenování aplikací OpenVAS podle závažnosti zranitelnosti

Zdroj: vlastní

Aplikace také zobrazí seznam nalezených zranitelností včetně závažnosti, zasaženého softwaru či operačního systému, popisu dané zranitelnosti s možným zneužitím útočníkem, řešením eliminace zranitelnosti a odkaz na databázi CVE. Jako příklad je uvedena zranitelnost s názvem *Ubuntu Update for apache2 USN-4113-1*, která má závažnost 7,8 z 10, je tedy vysoká. Mezi zasažený systém patří *apache2* balíčky na *Ubuntu 19.04, Ubuntu 18.04 LTS, Ubuntu 16.04 LTS*. Jedním z popisů této zranitelnosti je že komponenta z *mod_rewrite* v Apache byla v některých situacích zranitelná vůči přesměrování. Útočník by tento fakt mohl použít k odhalení citlivých informací nebo obcházení zamýšleného nastavení aplikace. Testovaná aplikace DVWA využívá ke svému chodu Apache, je tedy možné této zranitelnosti využít. Jako řešení této zranitelnosti je uvedena aktualizace *apache2* balíčku.

Dalším příkladem je zranitelnost nazvaná *phpinfo() output Reporting* se závažností 7,5. Mnoho instalačních tutoriálů PHP instruuje uživatele, aby vytvořil soubor nazvaný *phpinfo.ph*p nebo podobný, obsahující příkaz *phpinfo()*. Takový soubor je pak často ponechán v adresáři webového serveru. Z tohoto souboru může útočník získat informace jako je uživatelské jméno uživatele, který spouští proces PHP, zda je tento uživatel administrátor, IP adresa hostitele, verze webového serveru, operační systém včetně jeho verze a kořenový adresář webového serveru. Uvedeným řešením je smazání tohoto souboru ze serveru, nebo zákaz k jeho přístupu. Toto zjištění znamená, že server tento soubor obsahuje a útočník může bez práce zjistit informace uvedené výše a podle nabytých informací přizpůsobit následný útok.

3.9.2 Obrana

Obecně lze tento problém eliminovat častými aktualizacemi a sledováním nejnovějších nalezených chyb souvisejících s použitými komponentami. Vhodné je také odstranit nepoužité závislosti, nepotřebné funkce, součásti, soubory a dokumentaci [10]. Dalším možnost spočívá v implementaci periodicky se opakujícího skenování systému, touto funkcí aplikace OpenVAS disponuje [27]. Je možné si v ní nastavit například skenovaní pravidelně každý měsíc, stačí provést pár změn v nastavení vytvořeného skeneru. Aplikace také může posílat výsledky na email, takže se velká část práce automatizuje a výsledky pak stačí projít a provést navrhované řešení dané chyby.

3.10 Neošetřená přesměrování a předávání

Webové aplikace často přesměrovávají a předávají uživatele na jiné webové stránky a používají k určení cílové stránky nedůvěryhodné údaje. Bez řádného ověření mohou útočníci přesměrovat oběti na malwarem napadené stránky, aplikovat techniku phishingu (rybaření) nebo použít předání k získání přístupu k neoprávněným stránkám [10].

Útočník vytvoří odkaz na neověřené přesměrování z důvěryhodné stránky a přiměje oběť, aby na něj klikla. Jelikož odkaz původně směřuje na ověřenou stránku, oběť na něj pravděpodobně klikne. Útočník zacílí přesměrování tak, aby obešlo bezpečnostní kontroly. Aplikace často přesměrovávají uživatele na jiné stránky, nebo používají podobným způsobem vnitřní přesměrování v rámci aplikace. [21]

Občas může být cílová stránka specifikovaná v neověřeném parametru, čímž umožní útočníkovi, aby si vybral libovolnou cílovou stránku. Najít neověřené přesměrování na jiné stránky je jednoduché: hledejte přesměrování, která umožňují zadat celé URL. Najít neověřené přesměrování v rámci vnitřních stránek aplikace je těžší.

Tento druh přesměrování se může pokusit nainstalovat například škodlivý software nebo lstí přimět uživatele k tomu, aby prozradil heslo či jiné citlivé informace. Neověřené přesměrování může umožnit obejití dříve popsaných mechanismů řízení přístupu.

3.10.1 Praxe Security Shepherd (*Unvalidated Redirects and Forwards Lesson*)

Tento příklad úzce souvisí s kapitolou o CSRF útoku, který je popsaný výše. Účelem této lekce je obelhat administrátora, aby označil tuto lekci jako dokončenou. K tomu je využito Cross Site Request Forgery zranitelnosti, která je v aplikace nedostatečně ošetřena.

Jak je možné se dočíst z úvodního vysvětlení lekce, aplikace pouze kontroluje, zda je HTTP hlavička obou požadavků stejná.

Na obrázku níže, jsou všechny potřebné hodnoty barevně označené. Z těchto hodnot je potřeba poskládat odkaz, který přesměruje administrátora na jeho stránku pro udělení úspěšného dokončení lekce. Adresa byla sestavena následujícím způsobem: název barvy reprezentuje stejně barevně podtrženou část z obrázku níže.

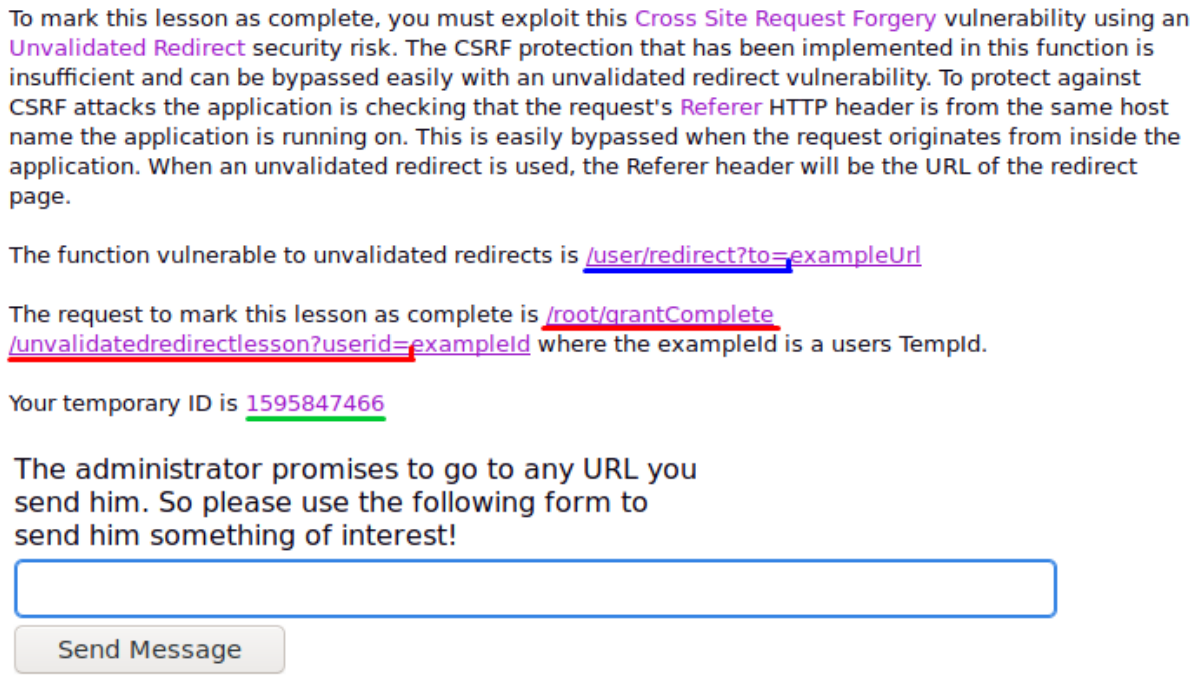
https://IPadresaModrýhttps://IPadresaČervenáZelená

Výsledná adresa po vytvoření vypadala takto:

```
1 https://127.0.0.1/user/redirect?to=https://127.0.0.1/root/grant-
  Complete/unvalidatedredirectlesson?userid=1595847466
```

Obrázek 26: Sestavená URL adresa pro přesměrování administrátora

Zdroj: vlastní



Obrázek 27: Zadání s vyznačenými údaji k sestavení adresy pro přesměrování

Zdroj: vlastní

3.10.2 Obrana

Jedním způsobem ochrany je přesměrování vůbec nepoužívat. Pokud je přesměrování použito, je důležité nepovolit URL adresu jako vstup uživatele pro výslednou adresu. Pokud nelze zabránit vstupu uživatele, je vhodná implementace kontroly, zda je dodaná hodnota platná a určená pro danou aplikaci. Pomoci může i vynutit, aby všechna přesměrování nejprve procházela stránkou, která upozorní uživatele, že se chystá opustit daný web, a jasně zobrazit cílovou stránku a nechat uživatele kliknutím potvrdit přesměrování na odkaz. [28]

Dalším způsobem je validovat vstup uživatele, k určení, zda je URL adresa bezpečná. Za tímto účelem je vhodné použít vestavěnou knihovnu nebo funkci, kvůli možnosti URL adresy analyzovat. Příkladem je funkce *parse_url()* v PHP.

Pro přehlednost a informovanost je nutné podotknout že vstup, který začíná požadovaným názvem domény není bezpečný. Také je nutné povolit pouze protokoly HTTP případně HTTPS a všechny ostatní protokoly, včetně URI *JavaScriptu*, jako je *javascript:alert(1)*, by měly být blokovány. [28]

ZÁVĚR

Cílem této bakalářské práce bylo otestování slabých míst v zabezpečení webové aplikace. Teoretická část cílila na seznámení se základními termíny potřebných pro lepší pochopení při jejich použití v praktické části práce. Tento úsek byl popsán v první kapitole, kde byly vysvětleny pojmy jako hrozby, dopady, rizika, aktiva a zranitelnosti. V této kapitole byly také popsány druhy bezpečnostních testů, jejich dělení a metody a postupy, které souvisí s bezpečnostními testy včetně fází penetračního testování.

Druhá kapitola představila: testovací aplikace Damn Vulnerable Web Application a OWASP Security Shepherd, dále nástroj Burp Suite a skener zranitelností OpenVAS.

Náplň třetí praktické kapitoly disponuje strukturou, kde na začátku každé podkapitoly byla charakterizována daná zranitelnost a způsob použití testovací aplikace, na které bylo následně demonstrováno, jak může útočník zranitelnost využít a jaké by to mohlo mít dopady. Kapitola aplikovala teoretické poznatky k objevení zranitelnosti a pokusu o její využití nekalými praktikami. Její realizace proběhla pomocí výše zmíněných testovacích aplikací a nástrojů. Důvodem, proč bylo využito vícero nástrojů, je, že každý z nich se zaměřuje na odlišné zabezpečení a na jiné zranitelnosti k otestování. Výběr zranitelností vyplývá ze seznamu nejkritičtějších zranitelností uvedených v přehledu OWASP Top Ten. Testované zranitelnosti zahrnují: injektování (SQL injection), chybnou autentizaci a správu relace, Cross-Site Scripting (XSS), nezabezpečený přímý odkaz na objekt, nezabezpečenou konfiguraci, expozici citlivých dat, chyby v řízení úrovní přístupů, Cross-Site Request Forgery (CSRF), použití známých zranitelností komponent, neošetřené přesměrování a předávání. Podkapitola každé zranitelnosti byla zakončena navrhovanou obranou. První odstavec byl zpravidla věnován obraně proti prováděnému útoku v praktické části této práce. Zbytek této podkapitoly byl psán více obecně, aby pokryl větší oblast z dané zranitelnosti. Navrhované obranné mechanizmy mnohdy obsahovaly odkaz na další studijní materiál zabývající se metodami obrany webové aplikace proti daným druhům útoků.

Na připravených testovacích aplikacích byly postupně představeny nejkritičtější zranitelnosti ve své základní podobě. V reálné situaci jsou aplikace většinou lépe zabezpečeny a jejich exploitace by tedy mohla vyžadovat hlubší znalosti a dovednosti. Nicméně tato bakalářská práce si kladla za cíl ukázat techniky spojené s běžnými bezpečnostními problémy v oblasti webových aplikací. Byly představeny připravené testovací aplikace a nástroje a pomocí nich ukázány postupy bezpečnostního testování. Aby nedošlo k porušení zákona č. 40/2009 Sb., byly pro testování vždy použity připravené testovací aplikace a nástroje.

POUŽITÁ LITERATURA

[1] Vybavenost domácností informačními a komunikačními technologiemi: Domácnosti s připojením k internetu. Český statistický úřad [online]. [cit. 2020-06-23]. Dostupné z: https://vdb.czso.cz/vdbvo2/faces/cs/index.jsf?page=vystup-objekt&pvo=ICT03B&z=T&f=TABULKA&skupId=2705&katalog=31031&pvo=ICT03B

[2] What Percentage of Internet Traffic Is Mobile? Oberlo [online]. [cit. 2020-06-23]. Dostupné z: https://www.oberlo.com/statistics/mobile-internet-traffic

[3] ČSN ISO/IEC 27005. *Informační technologie - Bezpečnostní techniky: Řízení rizik bezpečnosti informací.* První vydání. Praha: Českomoravské Normalizační Nakladatelství, 2019.

[4] ČSN EN ISO/IEC 27001. *Informační technologie - Bezpečnostní techniky: Systémy řízení bezpečnosti informací.* První vydání. Praha: Českomoravské Normalizační Nakladatelství, 2014.

[5] SELECKÝ, Matúš. *Penetrační testy a exploitace*. Brno: Computer Press, 2012. ISBN 978-80-251-3752-9.

[6] KIM, Peter. *Hacking: praktický průvodce penetračním testováním*. Brno: Zoner Press, 2015. Encyklopedie Zoner Press. ISBN 978-80-7413-313-8.

[7] STUTTARD, Dafydd a Marcus PINTO. *The web application hacker's handbook: finding and exploiting security flaws*. 2nd ed. Chichester: John Wiley, 2011. ISBN 978-1-118-02647-2.

[8] International Journal of Innovative Technology and Exploring Engineering (IJITEE): Web Application Penetration Testing [online]. Blue Eyes Intelligence Engineering & Sciences Publication, 2019, (8) [cit. 2020-02-18]. ISSN 2278-3075. Dostupné z: https://www.ijitee.org/wp-content/uploads/papers/v8i10/J91730881019.pdf

[9] About the OWASP Foundation. OWASP.org [online]. [cit. 2020-03-11]. Dostupné z: https://owasp.org/about

[10] OWASP Top 10 - 2013. OWASP.org [online]. 2013 [cit. 2020-03-11]. Dostupné z: https://owasp.org/www-pdf-archive/OWASP_Top_10_-_2013_Final_-_Czech_V1.1.pdf

[11] OWASP Top Ten. OWASP.org [online]. [cit. 2020-03-11]. Dostupné z: https://owasp.org/www-project-top-ten

[12] Damn Vulnerable Web Application (DVWA). DVWA [online]. [cit. 2020-06-19]. Dostupné z: http://www.dvwa.co.uk

[13] OWASP Security Shepherd. OWASP [online]. OWASP Foundation [cit. 2020-06-19]. Dostupné z: https://owasp.org/www-project-security-shepherd

[14] The Burp Suite family. PortSwigger [online]. United Kingdom: PortSwigger [cit. 2020-06-19]. Dostupné z: https://portswigger.net/burp

[15] Burp Suite documentation: contents. PortSwigger [online]. [cit. 2020-06-19]. Dostupné z: https://portswigger.net/burp/documentation/contents

[16] OpenVAS: About OpenVAS. OpenVAS [online]. Osnabrück Germany: Greenbone [cit. 2020-06-19]. Dostupné z: https://www.openvas.org/about.html

[17] DUŠEK, Petr. *Bezpečnost webových aplikací: Kurz cz.nic* [online]. 25. 4. 2019 [cit. 2020-03-20]. Dostupné z: https://www.silenceplease.cz/files/bwa_25_04_2019_cz_nic.pdf

[18] SQL Injection Prevention. OWASP Cheat Sheet Series [online]. OWASP, 2019 [cit. 2020-03-27]. Dostupné z: https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html

[19] Session Management. OWASP Cheat Sheet Series [online]. OWASP, 2019 [cit. 2020-04-29]. Dostupné z: https://cheatsheetseries.owasp.org/cheatsheets/Session_Management_Cheat_Sheet.html

[20] Cross Site Scripting (XSS). OWASP.org [online]. [cit. 2020-04-13]. Dostupné z: https://owasp.org/www-community/attacks/xss

[21] SULLIVAN, Bryan a Vincent LIU. *Web application security: a beginner's guide*. New York: McGraw-Hill, c2012. ISBN 978-0-07-177616-5.

[22] Rozdíl mezi HTTP a HTTPS? Víme bezpečně. PC PORADENSTVÍ.CZ [online]. [cit. 2020-03-22]. Dostupné z: http://www.pcporadenstvi.cz/rozdil-mezi-http-https-vime-bezpecne

[23] Password Storage. OWASP Cheat Sheet Series [online]. OWASP, 2019 [cit. 2020-05-03]. Dostupné z: https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html

[24] VALOIS, Mathieu, Patrick LACHARME a Jean-Marie Le BARS. Performance of Password Guessing Enumerators Under Cracking Conditions: ICT Systems Security and Privacy Protection - IFIP SEC. HAL archives-ouvertes [online]. 18.3.2019 [cit. 2020-05-04]. Dostupné z: https://hal.archives-ouvertes.fr/hal-02060091/file/ifipsec.pdf

[25] Cross Site Request Forgery (CSRF). OWASP.org [online]. [cit. 2020-03-22]. Dostupné z: https://owasp.org/www-community/attacks/csrf

[26] Cross-Site Request Forgery. OWASP Cheat Sheet Series [online]. OWASP, 2019 [cit. 2020-05-03]. Dostupné z: https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html

[27] Greenbone Security Manager with Greenbone OS 6: User Manual. Greenbone [online]. Osnabrück Germany: Greenbone Networks [cit. 2020-06-18]. Dostupné z: https://docs.greenbone.net/GSM-Manual/gos-6/en/

[28] Unvalidated Redirects and Forwards. OWASP Cheat Sheet Series [online]. OWASP, 2019 [cit. 2020-05-03]. Dostupné z: https://cheatsheetseries.owasp.org/cheatsheets/Unvalidated_Redirects_and_Forwards_Cheat_Sheet.html#preventing-unvalidated-redirects-and-forwards

PŘÍLOHY

Příloha A – Konfigurace virtuálního počítače s Kali Linuxem

Hardwarová konfigurace virtuálního počítače:

- 4 jádra procesoru
- 4096 MB RAM
- 40 GB diskového prostoru
- Režim sítě nastaven na NAT

Softwarové vybavení virtuálního počítače:

- Kali GNU Linux Rolling 2020.2
- Mozilla Firefox 68.10.0
- Chromium 83.0.4103.116
- Apache 2.4.43
- PHP 7.4.5
- Java 11.0.7
- Apache Maven 3.6.3
- BurpSuite Community Edition 2020.6
- Greenbone Security Assistant 7.0.3
- OpenVAS Scanner 5.1.3

Příloha B – Konfigurace virtuálního počítače s Ubuntu

Hardwarová konfigurace virtuálního počítače:

- 4 jádra procesoru
- 2048 MB RAM
- 40 GB diskového prostoru
- Režim sítě nastaven na NAT

Softwarové vybavení virtuálního počítače:

- Ubuntu 18.04.1 LTS, 64-bit edice
- Mozilla Firefox 72.0.1
- Apache 2.4.29
- MariaDB 10.1.29
- PHP 7.2.3
- Apache Maven 3.6.3
- DVWA 1.9
- OWASP Security Shepherd 3.1