Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačního inženýrství

Bakalářská práce

Bezpečnostní testování webových aplikací

Šárka Möstlová

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Mgr. Šárka Möstlová

Systémové inženýrství a informatika
Informatika

Název práce

Bezpečnostní testování webových aplikací

Název anglicky

Security Testing of Web Applications

Cíle práce

Práce se zabývá problematikou bezpečnostního testování webových aplikací. Hlavním cílem práce je poskytnout přehled aktuálních bezpečnostních rizik, jakým jsou webové aplikace nejčastěji v dnešní době vystaveny, a prakticky demonstrovat jednotlivá bezpečnostní rizika prostřednictvím testování konkrétní webové aplikace. V rámci nálezů v podobě identifikovaných rizik bude formulováno doporučení, jak daná pochybení odstranit, popř. jim předcházet.

Metodika

Bakalářská práce sestává ze dvou částí – teoretické a praktické. Metodiky zpracování teoretické části spočívá ve studiu odborných informačních zdrojů. Na základě syntézy zjištěných poznatků budou formulována teoretická východiska práce. Bude poskytnut přehled aktuálních bezpečnostních rizik, na jejichž základě bude navržen postup praktickou část.

V praktické části bude navržen postup testování a následně budou provedeny demonstrace bezpečnostního testování zaměřené na možnost využití bezpečnostních rizik popsaných v teoretické části. Jednotlivé testy budou vyhodnoceny, jejich výsledky popsány a budou navrženy možnosti eliminace daného rizika v rámci testované aplikace.

Poznatky a výstupy z teoretické i praktické části budou na závěr shrnuty.

Doporučený rozsah práce

30-40 stran

Klíčová slova

bezpečnostní testování, security testing, owasp, webová aplikace, bezpečnostní riziko, penetrační testování

Doporučené zdroje informací

KANER, C. – BACH, J. – PETTICHIRD, B. Lessons learned in Software Testing. New York: WILEY, 2001. ISBN 0-471-08112-4.

KOFLER, M. Hacking & Security, Das umfassende Buch. Bonn: Rheinwerk Verlag, 2018. ISBN 978-3-8362-4548-7.

KUROSE, J F. – ROSS, K W. *Počítačové sítě.* Brno: Computer Press, 2014. ISBN 978-80-251-3825-0.

MARSHALL, J. Hands-On Bug Hunting for Penetration Testers. Brimingham: Packt, 2018. ISBN 978-1-78934-420-2.

OWASP Top Ten Project – OWASP. [online]. Dostupné z: https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project.

SEITZ, J. Hacking mit Python, Fehlersuche, Programmanalyse, Reverse Engineering. Heidelberg: dpunkt.verlag, 2009. ISBN 978-3-89864-633-8.

Předběžný termín obhajoby

2019/20 LS – PEF

Vedoucí práce

Ing. Jiří Brožek, Ph.D.

Garantující pracoviště

Katedra informačního inženýrství

Elektronicky schváleno dne 19. 2. 2020

Ing. Martin Pelikán, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 19. 2. 2020

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 11. 10. 2020

Čestné prohlášení

Prohlašuji, že svou bakalářskou práci "Bezpečnostní testování webových aplikací" jsem vypracoval(a) samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor(ka) uvedené bakalářské práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 30. listopadu 2020 _____

Poděkování

Rád(a) bych touto cestou poděkoval(a) vedoucímu Ing. Jiřímu Brožkovi, Ph.D. za odborné vedení a podporu během vypracování práce.

Bezpečnostní testování webových aplikací

Abstrakt

Práce se zabývá bezpečnostním testováním webových aplikací. Na úvodních stránkách dochází k definování pojmosloví v oblasti testování, IT rizik a bezpečnosti. Co všechno lze do bezpečnostních ověření a testů zařadit, je ukázáno na průchodu bezpečným vývojovým cyklem softwaru. Krátce jsou představena i specifika bezpečnostních testů v agilním vývoji a DevSecOps. Následuje souhrn poznatků ohledně nejčastějších rizik, které hrozí webovým aplikacím, a jak jsou konkrétní slabiny a zranitelnosti evidovány a klasifikovány. Pro provedení testů je dále nutné představit metodologie testování a konkrétně projít jednotlivé kroky při přípravě, provedení testů a reportování. Na základě těchto poznatků je autorkou práce provedeno testování jí vytvořené webové aplikace. Jedná se o testy ve fázi implementace a verifikace v rámci vývojového cyklu softwaru. Identifikované bezpečnostní nálezy jsou opatřeny doporučeními pro odstranění chyb.

Klíčová slova: bezpečnostní testování, security testing, OWASP, webová aplikace, bezpečnostní riziko, penetrační testování, bezpečnostní slabiny, zranitelnosti IT systému, DevSecOps

Security Testing of Web Applications

Abstract

This thesis focuses on security testing of web applications. On the introductory pages, the terminology in the field of testing, IT risks, IT security, web security testing is defined. In the description of the secure life cycle development, different kinds of security verifications and tests are described. The specifics of security activities in agile development and DevSecOps are also briefly introduced. Summary of knowledge about the most common web security risks follows and chosen standardized systems of enumerations of vulnerabilities and weaknesses are included as well. In order to perform the tests, it is necessary to define the testing methodology and specifically explain different stages from the test preparation, test execution to bug reporting. Based on the theoretical grounds a web application created by the author of the thesis is being tested according to chosen methodologies. Both tests from the implementation and verification phase of a secure software development lifecycle are included. Identified vulnerabilities are explained and described accompanied by recommendations to eliminate those security bugs.

Keywords: Security Testing, OWASP, Web Application, Security Risk, Penetration Testing, Security Weakness, IT System Vulnerability, DevSecOps

Obsah

1 Úvod ... 10

2 Cíl práce a metodika ... 11

2.1 Cíl práce ... 11

2.2 Metodika ... 11

3 Teoretická východiska ... 13

3.1 Bezpečnostní testování ... 13

3.1.1 Testování bezpečnosti, rizika a IT bezpečnost ... 13

3.1.2 Bezpečnostní testování ve vývojovém cyklu softwaru ... 14

3.1.2.1 Fáze vývojového cyklu ... 14

3.1.2.2 Bezpečnostní specifika agilního vývoje ... 19

3.1.2.3 DevSecOps ... 20

3.2 Zranitelnosti webových aplikací ... 21

3.2.1 Architektura webových aplikací ... 21

3.2.2 Zranitelnosti, zneužití a jejich evidence ... 22

3.2.2.1 Nejčastější rizika podle OWASP Top Ten ... 23

3.2.2.2 Zranitelnosti ve webových frameworcích ... 28

3.3 Metodologie testování webových aplikací ... 28

3.3.1 Příprava testů ... 29

3.3.2 Provedení testů ... 30

3.3.3 Reporting a doporučení ... 34

4 Vlastní práce ... 36

4.1 Testovaná aplikace ... 36

4.1.1 Funkční specifikace aplikace ... 36

4.1.2 Technická specifikace ... 38

4.2 Příprava testů a nástrojů ... 39

4.3 Provedení testu a výsledky ... 39

4.3.1 Testování bezpečnosti zdrojového kódu ... 39

4.3.2 Bezpečností testy ve fázi verifikace ... 41

5 Zhodnocení a doporučení ... 50

5.1 Zhodnocení provedených testů ... 50

5.1.1 Doporučení pro nápravu v oblasti nedostatků zdrojového kódu ... 51

5.1.2 Doporučení k odstranění nálezů ve fázi verifikace ... 52

6 Závěr ... 57

7 Seznam použitých zdrojů ... 59

8 Přílohy 65
8.1 Příloha č. 1: Seznam bezpečnostních nálezů v testované aplikaci 66
8.2 Příloha č. 2: CD se zdrojovým kódem 66

Seznam obrázků

Obrázek 1: Bezpečnostní aktivity vývojového cyklu podle Microsoft 15
Obrázek 2: Přihlašovací stránka testované aplikace 36
Obrázek 3: Stránka s výběrem rozhodnutí 37
Obrázek 4: Potvrzující stránka 37
Obrázek 5: Kontaktní formulář 38
Obrázek 6: Projekt na GitHub s uloženým zdrojovým kódem 38
Obrázek 7: Náhled na výsledek analýzy SonarQube 40
Obrázek 8: Náhled na nalezenou zranitelnost v SonarQube 40
Obrázek 9: Uživatelské rozhraní OWASP ZAP a Spider 42
Obrázek 10: Burp Suite se zobrazením řádky, kde došlo k prolomení hesla 45
Obrázek 11: Demonstrace zranitelnosti na XSS útok 48
Obrázek 12: Spuštění testu v Sqlninja 49
Obrázek 13: Změna hodnot v databázi u všech uživatelů (E-mail) 49
Obrázek 14: Chybová hláška prozrazující informace o databázi 50

1 Úvod

Podle zprávy NTT Global Threat Intelligence Report z roku 2019 představují útoky proti webovým aplikacím více jak 32% všech hrozeb, nejvíce pak v sektorech finančnictví, obchodu a profesních službách nebo zdravotnictví. Útoky na webové aplikace se zdvojnásobily v posledních letech. Rok 2018 byl podle uvedeného reportu rekordním v počtu nalezených nových zranitelností[1].

Během šíření onemocnění Covid-19 došlo k většímu tlaku na podniky v oblasti digitální transformace – firmy se snaží podporovat své pracovníky v práci na dálku. Tímto krokem však otvírají nové přístupové cesty útočníkům a nejsou ještě připraveni je chránit. Výsledkem je ztráta dat, zisku nebo reputace. Podle reportu Hacker One zaznamenali dotazovaní lídři v oblasti bezpečnosti 30% nárůst útoků od začátku pandemie a zároveň 64% z nich očekává, že se stanou v této době pravděpodobněji obětí úniku dat[2].

Toto ale není téma, které je běžnému uživateli vzdálené. V dnešní době není takřka možné se při každodenních činnostech či během vykonávání pracovních aktivit vyhnout kontaktu s digitálním světem, potažmo s aplikacemi přístupnými pouze přes internet. Aplikace přístupné přes internet jsou vystavené řadě bezpečnostních rizik. Dochází k neustálému předávání velkého množství dat a jak uživatelé, tak provozovatelé služeb prostřednictvím webových aplikací mají zájem na tom, aby takové předávání bylo bezpečné.

Je důležité porozumět aktuálním rizikům a hrozbám i způsobu, jak zranitelnosti webových aplikací v rámci vývoje softwaru vznikají. To, zda jdou aplikace zralé na fungování v produkčním prostředí bez zásadních chyb v oblasti zabezpečení, lze ověřit během bezpečnostního testování, které je ve vztahu k webových aplikacím ústředním tématem této práce.

[1]NTTSecurity, 2019.
[2] HackerOne, 2020.

2 Cíl práce a metodika

2.1 Cíl práce

Hlavním cílem této práce je poskytnout přehled aktuálních bezpečnostních rizik, jakým jsou webové aplikace nejčastěji v dnešní době vystaveny a prakticky demonstrovat jednotlivá bezpečnostní rizika prostřednictvím testování konkrétní webové aplikace. V rámci nálezů v podobě identifikovaných rizik bude formulováno doporučení, jak daná pochybení odstranit, popř. jim předcházet.

Konkrétně půjde o splnění následujících dílčích cílů:

1. Definování bezpečnostního testování a jeho místa ve vývojovém cyklu softwaru
2. Charakteristika nejčastějších bezpečnostních rizik u webových aplikací
3. Zjištění aktuálních způsobů a metodologií bezpečnostního testování včetně nástrojů
4. Zajištění objektu testování (webové aplikace) a představení funkční a technické specifikace
5. Příprava testování a vymezení nástrojů pro provedení testů
6. Provedení testů
7. Vyhodnocení nálezů a formulace závěru o stavu testované aplikace
8. Formulace doporučení a opatření k odstranění reportovaných nálezů
9. Celkové zhodnocení

2.2 Metodika

Bakalářská práce sestává ze dvou částí – teoretické a praktické. Metodika zpracování teoretické části spočívá ve studiu odborných informačních zdrojů se zaměřením na aktuální dostupnou literaturu odrážející nejnovější poznatky a trendy. Na základě syntézy zjištěných poznatků budou formulována teoretická východiska práce. Bude poskytnut přehled pojmosloví v této oblasti a ukázáno místo bezpečnostního testování napříč vývojovým cyklem softwaru, včetně specifik agilního vývoje. Dále bude poskytnut přehled aktuálních bezpečnostních rizik a uvedeny mezinárodní standardy pro evidenci slabin a zranitelností. Budou představeny způsoby testování a vybrané metodologie, na jejichž základě bude navržen postup pro praktickou část.

V praktické části bude testována jednoduchá webové aplikace vytvořená pro tento účel přímo autorkou práce. V rámci tvorby kódu na straně backendu nebude využito žádných dostupných frameworků.

Dále dojde k zajištění testovacího prostředí a testovacích nástrojů pro testy. Je potřeba se seznámit se způsobem používání vybraných nástrojů – zejm. SonarQube, OWASP ZAP, Burp Suite. Jedna část testů se zaměří na testování za znalosti vnitřních souvislostí fungování aplikace (za znalosti zdrojového kódu), a bude směřovat do fáze vývoje webové aplikace. Další část testů se zaměří na testování na konci vývojového cyklu softwaru – ve fázi verifikace. Žádoucí je provedení jak automatizovaných, tak manuálních testů.

Testy se svým zaměřením soustředí na nejčastější zranitelnosti a rizika a zároveň budou sledovat vybranou metodologii, která známé zranitelnosti reflektuje. Jednotlivé testy budou vyhodnoceny a jejich výsledky popsány v podobě bezpečnostních nálezů ohodnocených závažností. Zásadní část práce bude představovat formulace konkrétních a praktických opatření a doporučení k eliminaci daného rizika v rámci testované aplikace.

3 Teoretická východiska

3.1 Bezpečnostní testování

Bezpečnostní testování webových aplikací je slovní spojení, které si budeme pro účely této práce v následujících kapitolách a podkapitolách blíže definovat. Ve světě IT bezpečnosti existuje řada směrnic a metodologií, podle kterých by rozbor i jen jednoho jednotlivého pojmu vydal na samostatnou publikaci. Následující vymezení tedy tvoří pouze autorkou vybrané minimum pro uchopení tématu a praktické využití ve vlastní části práce.

3.1.1 Testování bezpečnosti, rizika a IT bezpečnost

Testováním softwaru rozumíme způsob posouzení kvality softwaru a sníženi rizik selhání softwaru v provozu. Cíle testování se pak liší v závislosti na kontextu testovaného systému, úrovni testu a také modelu životního cyklu vývoje (ISTQB, 2018, str. 13). Způsobů, jak kvalitu softwaru posoudit, je mnoho a liší se i jejich cíle.

Cíle bezpečnostních testů jsou založeny na **bezpečnostních rizicích**, přičemž za riziko lze považovat míru, do jaké je entita ohrožena potenciální okolností nebo událostí, kdy vyhodnocujeme nepříznivé dopady, které by vznikly, kdyby daná událost nastala, a také pravděpodobnost výskytu. **Rizika v informační bezpečnosti** vyplývají ze ztráty důvěrnosti, integrity nebo dostupnosti informací nebo informačních systémů a odrážejí potenciální dopady na provoz organizace (např. funkce, reputace), majetek, jednotlivce či jiné organizace a země (Rice a kol., 2016, str. 12; Ross[3], 2012, str. 15). Existující rizika jsou nezbytným předpokladem pro bezpečnostní testy. Za **IT bezpečnost** můžeme tedy označit stav, ve kterém jsou výše uvedené hodnoty jako důvěrnost, integrita a dostupnost informací a informačního systému prostřednictvím vhodných opatření chráněna (Simon, 2019, str. 9).

V **bezpečnostních testech webové aplikace** se zaměřujeme na vyhodnocení zabezpečení webové aplikace, které zahrnuje aktivní analýzu aplikace s ohledem na slabiny, technické chyby a zranitelnosti (OWASP, 2013, str. 29). Na různé druhy

[3] Zmíněný NIST Special Publication 800-30 je významný pro severoamerický trh, obecně platný je dokument ISO 31000, který popisuje řízení rizik definované v obecné rovině a nasaditelné do praxe organizací.

bezpečnostních testů a opatření k zajištění bezpečnosti se podíváme v následujících kapitolách, vyplývají i ze svého umístění ve vývojovém cyklu softwaru.

3.1.2 Bezpečnostní testování ve vývojovém cyklu softwaru

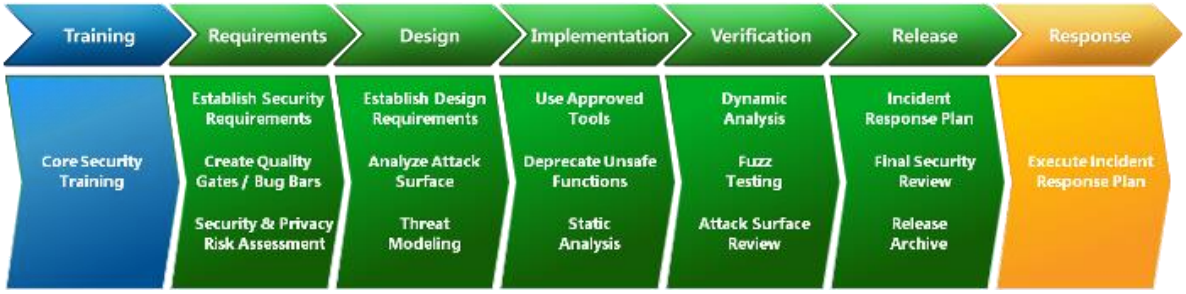
Stejně jako testování softwaru obecně má nepopiratelné místo v rámci vývojového cyklu softwaru, ale bývá často nesprávně prováděno v samotném závěru (takřka po ukončení) vývojového cyklu, kdy už je software hotov, tak i na témata bezpečnosti lze myslet dříve, než jen jako drobnou, pro forma, položku před nasazením systému do produkce. Jako nejlepší metodu prevence bezpečnostních chyb v produkčních aplikacích je vylepšit vývojový cyklus softwaru (SDLC – Software Development Life Cycle) tím, že zahrneme bezpečnost do každé jeho fáze: do definování záměru (define), návrhu (design), vývoje (develop), nasazení (deploy) a údržby (maintain) (Meucci, 2013, str. 12).

3.1.2.1 Fáze vývojového cyklu

V praxi existuje více modelů vývojového cyklu a jednotlivé blíže definované aktivity se budou lišit u iterativních[4] a vodopádových[5] modelů. Výběr modelu závisí na způsobu organizace, konkrétním projektu, jeho rozsahu atd. Standardy k životnímu cyklu softwaru nalezneme v dokumentu ISO 12207, který představuje mezinárodní normu definující standardní proces životního cyklu software, nebo ISO 15288 zabývající se procesy a fázemi životního cyklu. Jako příklad modelu, který staví téma bezpečnosti jako ústřední aspekt, bývá uváděn Microsoft Software Development Lifecycle, dále jen SDL (Simon, 2019, str. 163). Jako inspirace mohla do května 2019 také sloužit publikace NIST[6] zaměřená na bezpečnostní aspekty ve vývojovém cyklu[7], která byla nahrazena komplexnější sadou dokumentů[8] (Ross, 2019).

[4] V iterativním vývoji jsou funkce specifikovány, navrhovány, vyvíjeny a testovány v sérii cyklů často o pevné délce trvání. Každý cyklus představuje rostoucí podmnožinu celkové sady funkcí postupně dodávané dokud není dodán celý software (ISTQB, 2018, str. 29).
[5] Ve vodopádovém modelu jsou aktivity vývoje dokončovány postupně jedna po druhé a testování následuje až když je celý vývoj ukončen (ISTQB, 2018, str. 28).
[6] National Institute of Standards and Technology – americký Národní institute pro standardy a technologii.
[7] Blíže viz Kissel, 2008.
[8] Jmenovitě jde krom popsaných ISO-dokumentů o: Vývoj systémů odolných vůči kybernetickým útokům: přístup systémového bezpečnostního inženýrství; Systémové bezpečnostní inženýrství - úvahy o multidisciplinárním přístupu při konstrukci důvěryhodných zabezpečených systémů; Rámec řízení rizik pro informační systémy a organizace - přístup životního cyklu systému k zabezpečení a ochraně osobních údajů (Ross, 2019).

Microsoft SDL doporučuje povinně využívat pravidla tohoto cyklu v případech, kdy se jedná o vývoj softwaru určeného pro obchodní prostředí, software, kde se zpracovávají osobně identifikovatelné informace nebo jiné citlivé informace anebo dochází ke komunikaci přes internet nebo jiné sítě. Bezpečnostní aktivity zařazené do vývojového cyklu znázorňuje následující obrázek. (Microsoft, 2012).



Obrázek 1: Bezpečnostní aktivity vývojového cyklu podle Microsoft

Trénink

Trénink/školení je průřezová aktivita, která se netýká jen vývojářů, ale i servisních techniků, projektových a produktových manažerů atd. Microsoft doporučuje absolvovat takový trénink jednou ročně pro vývojáře. Tematicky se jedná o povědomí v oblasti bezpečného vývoje (redukování ploch útoku, hloubková ochrana, zásada nejmenších privilegií atd.), modelování hrozeb, bezpečnostního kódování (ochrana proti cross-site scriptingu, SQL injekci atd.), bezpečnostního testování (odhad rizik, metodologie testování a automatizace atd.), ochrany osobních údajů (Microsoft, 2012, str. 14).

Za zmínku stojí nová verze Rámce bezpečnostních znalostí (Security Knowledge Framework) od OWASP, která poskytuje vývojářům možnost cvičit se ve psaní bezpečného kódu, obsahuje tzv. checklisty pro různé oblasti implementace bezpečnostních požadavků, testovací laboratoře atd. (SKF, 2020).

Definování záměru – požadavky

Zde je doporučováno pro **určení bezpečnostních požadavků** připravit dotazník za účelem zjištění, zda bude vývojový tým podléhat pravidlům SDL. V případě že ano, je určen bezpečnostní konzultant/poradce, který pomůže identifikovat bezpečnostní rizika na základě vyhodnocení technických otázek v rámci bezpečnostního **posouzení rizik**[9], dále

[9] Security Risk Assessment.

působí jako kontaktní bod pro finální bezpečnostní kontrolu[10] (viz dále). Vymezuje se také, jaké částí projektu budou potřebovat model hrozeb, review bezpečnostního designu, užití penetračních testů, definování požadavků na fuzz testing – testování zabezpečení používaná k odhalení bezpečnostních slabin zadáváním velkého množství náhodných dat do testované komponenty (ISTQB, 2018, str. 54). Předmětem zájmu v této fázi je také definování dopadu do oblasti ochrany osobních údajů.

Vytváří se také systém pro vyhodnocení zralosti projektu a stanovení standardů kvality, co do splnění bezpečnostních požadavků v podobě **definování bezpečnostních bran**[11] či tzv. projektových bezpečnostních bug bars - laťky požadavků, ze kterých nelze slevit, ani v čase blížícího se releasu (Microsoft, 2012, str. 17).

Návrh softwaru

V této fázi jde o stanovení požadavků na design, vymezování funkční a technické specifikace, na které může navázat **mapování attack surface** – ploch útoku. Spíše než o počítání chyb na straně kódu, počítáme příležitosti útoku. Autoři popisují 3 dimenze: cíle a aktivátory, kanály a protokoly a přístupová práva. Čím více je plocha systému exponovaná, tím více příležitostí k útoku nacházíme a tím více se může systém stát cílem útoku. Snížením ploch útoku na systém tedy zvyšujeme bezpečnost systému (Howard, 2003).

Významnou bezpečnostní aktivitu zde představuje také **modelování hrozeb**[12]. Jde o používání modelů k najití bezpečnostních problémů a návrhu jejich zmírnění. Zjednodušeně řečeno se tak děje při nalézání odpovědi na otázky: co vyvíjíme, co se může pokazit a co můžeme učinit s věcmi, které se mohou pokazit. Při definování prvků systému, které ponesou potenciální chyby, uvádí autoři metodu STRIDE zaměřenou na šest kategorií bezpečnostních hrozeb:

Spoofing – podvržení identity uživatele. Požadovaná vlastnost, kterou systém musí mít je dostatečný způsob autentizace.

Tampering – neoprávněná modifikace produktu (např. paketů při přenosu). Požadovaná vlastnost hodná sledování je integrita (opatřením by např. bylo použít https/ssl).

[10] V angličtině Final Security Review (FSR).
[11] V angličtině Quality Gates (QG).
[12] V angličtině Threat Modeling.

Repudiation – odmítnutí akce. Jde o navržení systému tak, aby docházelo k logování a logy byly chráněny, např. užitím spolehlivého transportu logů.

Information disclosure – vyzrazení informace. Sledovaným prvkem je zde důvěrnost.

Denial of Service – odepření služby. Je potřeba zajistit aby systém byl dostupný.

Elevation of Privilege – zde se jedná o problém chybějící autorizace (Shostack, 2014, str. 22).

Vývoj – implementace

Z bezpečnostního hlediska je ve fázi vývoje cílem detekovat a odstranit bezpečnostní chyby nejlépe ze začátku vývojového cyklu. K tomu může sloužit řada nástrojů. Microsoft představuje řadu bezpečnostních požadavků (nastavení atributů cookies, vhodné používání knihoven, zakázaná API, kompilování nativního kódu, bezpečné metody přístupu do databází, používání HTTPS, používání schválených nástrojů atd.) a požadavků v oblasti ochrany osobních údajů (Microsoft, 2012, str. 44).

Včasnou detekci chyb umožňuje systematická **kontrola kódu** (code review). Cílem kontroly kódu je identifikovat bezpečnostní chyby aplikace související s jejími funkcemi společně s přesnými příčinami. Se vzrůstající komplexitou aplikací již nestačí klasické formy testování, ale je nutné porozumět kódu aplikace včetně externích komponent. Organizace, které začlenily kontrolu kódu do vývojového cyklu, produkovaly z hlediska bezpečnosti pozoruhodně lepší kód (OWASP, 2017).

Organizace OWASP[13] vytvořila samostatný, volně dostupný, dokument Průvodce kontrolou kódu - OWASP Code Review Guide, který rozdílně od klasické kontroly kódu bere v potaz i bezpečnost. Na rozdíl od penetračních testů (viz dále), které jsou obecně spíše black box testem (tzv. test černé skříňky)[14], se zde jedná o techniku testování za znalosti vnitřní struktury testované aplikace[15]. Kontrola kódu může být prováděna manuálně, ale také automatizovaným nástrojem. Firmy využívají jak interní, tak externí, komerční i open source nástroje pro provedení **statické analýzy kódu**. Nevýhodou automatizovaných nástrojů je skutečnost, že neodhalí chyby v logice aplikace, jsou často

[13] Organizace OWASP (Open Web Application Security Project) je světová otevřená komunita zaměřená na zlepšování bezpečnosti aplikačního softwaru, jejíž cílem je zviditelnit témata aplikační bezpečnosti, tak aby organizace a jednotlivci mohli činit informovaná rozhodnutí o rizicích v této oblasti (OWASP, 2017, str. 220).

[14] Test provedený bez znalosti vnitřní struktury systému, zaměřený na vstupy a výstupy testovaného objektu (ISTQB, 2019, str. 56).

[15] Tzv. test bílé skříňky - white box test.

zaměřené jen na některé programovací jazyky, těžko identifikují chyby v návrhu a také ne vždy označí správně identifikovanou chybu, která chybou být nemusí[16] (OWASP, 2017).

Ověření - testování

Ve fázi ověření jde o zajištění, že kód splňuje zásady zabezpečení a ochrany osobních údajů zejména prostřednictvím testování. Testování má začít bezprostředně potom, co je kód napsán (Microsoft, 2012, str. 44).

Vezmeme-li v úvahu 4 úrovně testů – test komponent[17], integrační testy[18], systémové testy[19] a akceptační testy[20], je potřeba poukázat na skutečnost, že možnosti, jakými je systém činěn zranitelným, nepředstavují jen jednoduchý součet zranitelností daných komponent integrovaných do systému, nýbrž díky interakci mezi komponentami a většími systémy se objevují vektory útoku, které při testování samotných komponent nejsou zřejmé. Bezpečnostní testy mají tedy své místo od komponentově-integrační úrovně, přes systémovou až po akceptační úroveň testů. Cílem bezpečnostních akceptačních testů je ověření bezpečnostních akceptačních kritérií z perspektivy uživatele, popř. potenciálního útočníka (Simon, 2019, str. 198).

Microsoft popisuje nutnost re-evaluovat **plochy útoku** (attack surface) a související dokumentaci, popisuje také **testovací aktivity**, které bývají někdy přiřazovány do kategorie **penetračních testů** (Microsoft, 2012, str. 53). Penetrační test lze popsat jako rozsáhlý bezpečnostní test počítačového systému. Často je k provedení takového testu pověřena osoba nebo organizace odlišná od zadavatele. Pentesteři pak zkouší napadnout systém a nalézt bezpečnostní mezery. Nálezy nejsou zneužity, ale nahlášeny zadavateli, aby byly opraveny (Kofler a kol., 2019, str. 23).

Jedním druhem testování v této fázi vývojového cyklu je **fuzz testing**. Původně byla tato metoda uveřejněna roku 1988 jako projekt předložený na Wisconsin-Medison

[16] Tzv. false positive.
[17] Test komponent, také unit test, nebo test modulu, se zaměřuje na prvky systému, které jsou samostatně testovatelné a je možné je provádět izolovaně od zbytku systému (ISTQB, 2019, str. 31).
[18] Integrační test se zaměřuje na interakci mezi komponenty a systémy. Rozlišujeme 2 úrovně integračních testů: komponentově-integrační (integrace mezi komponenty) a systémově-integrační testy (SIT – rozhraní systémů, interakce mezi balíky, mikroservisy a také interakce s externími rozhraními a organizacemi atd.), viz ISTQB, 2019, str. 32).
[19] Systémový test se zaměřuje na chování a vlastnosti celého systému či produktu, často pojímaný jako end-to-end test a ověřovány jsou také ne-funkční aspekty testovaného systému (ISTQB, 2019, str. 34).
[20] Akceptační test se zaměřuje na systém jako celek s cílem vytvořit důvěru v kvalitu systému jako celku a ověřit, zda funguje dle očekávání. Častou formou je UAT – User Acceptance Testing, kde výše uvedené verifikuje zamýšlený uživatel (ISTQB, 2019, str. 36).

univerzitě, odboru počítačových věd. Šlo o ověření robustnosti různých obslužných systémů UNIX (př. vi, mail, awk), přičemž se vytvořil generátor fuzzů – náhodného proudu znaků - s cílem „rozbít“ testovaný systém. Poté došlo k analýze, které vstupy narušení systému způsobily (Miller, 1988, str. 1). Přesto, že fuzzing je jedna z nejběžnějších metod používaných hackery k nalezení zranitelností v systému, platí, že odhalí zpravidla jen jednodušší chyby a aby byl proveden efektivně, vyžaduje to značný čas. Moderních nástrojů na fuzz testing existuje mnoho, př.: OWASP WSFuzzer, Peach Fuzzer, Spike Proxy, Webscarab (Kothe, 2019).

Nasazení a údržba

Ke konci vývoje je potřeba finálně ověřit, že software je dostatečně bezpečný pro nasazení. K tomu sloužívá **finální bezpečnostní review** (Final Security Review - FSR), kde se kontroluje, že software je v souladu s původními a dodatečně formulovanými požadavky. Jsou revidovány bezpečnostní modely, validovány bezpečnostní nástroje, odstraněny nalezené zranitelnosti (Microsoft, 2012, str. 59).

Cílem bezpečnostních aktivit po nasazení softwaru do provozu je najít slabiny, které se objeví teprve během provozu nebo během údržby systému. Příčinou takových nálezů může být nedostatečné testování změn softwaru, implementované opravy jiných chyb. Doporučuje se zařadit do testování i **bezpečnostní regresní testy**[21], při kterých testujeme, zda nasazené změny neovlivní negativně bezpečnostní vlastnosti systému. I v této fázi hrají roli **penetrační testy**, protože se zpravidla provádějí až ve fázi, kdy je testovaný systém takřka hotov anebo se nachází v provozu (Simon, 2019, str. 202).

3.1.2.2 Bezpečnostní specifika agilního vývoje

Bezpečnostní vývoj podle **agilní metodiky**[22] nese svá specifika. Těžištěm je zde tzv. sprint (opakující se běh trvající 15 až 60 dnů), během kterého je sada funkcí nebo user stories navržena, vyvinuta, testována a potenciálně dodána zákazníkovi. Před zahájením

[21] Regresní testy ověřují, zda nasazená změna v jedné části kódu (ať už oprava chyby nebo jiná změna) neovlivní chování jiné části kódu stejného, či souvisejícího systému. Regresní testy jsou vhodným kandidátem pro automatizaci (ISTQB, 2019, str. 41).

[22] Dnešní agilní metodika, vychází z agilního manifestu vývoje uveřejněného roku 2001, zaměřeného na tehdy nový způsob vývoje software: rychlé dodání software v řádu týdnů až měsíců za spolupráce týmu s businessem, za upřednostňování rychlé reakce na změny před dodržováním plánu, fungujícího software před vyčerpávající dokumentací, interakce před procesy a nástroji atd. (Beck a kol., 2001).

se vybírá ze seznamu požadovaných funkcionalit (tzv. backlog). Požadavky SDL jsou zde reprezentovány jako úkoly a přidávány do sprintu (v podobě ne-funkčních požadavků).

Bezpečnostní požadavky se dělí do 3 skupin:

1) Požadavky, které se provádí **každý sprint** (např. využití automatizovaných nástrojů – statická analýza kódu).
2) Požadavky, které se naplňují **pravidelně během vývoje**[23]. Sem patří aktivity spadající do tří skupin: ověřovací úkoly (např. analýza ploch útoku, testování vstupů - input validation testing), kontrola návrhu (např. kontrola návrhu šifrování, přidělení minimálních oprávnění) a plánování (např. aktualizace laťky požadavků - bug bar).
3) Požadavky plněné **jedinkrát během vývojového cyklu**: požadavky, které je potřeba naplnit, např. zpravidla ze začátku vývoje (vytvoření modelu hrozeb, vybrání bezpečnostního poradce apod.).

Problematicky se jeví pokus zařadit všechny potřebné požadavky do krátkých sprintů. Jistým ulehčením je rozdělení úkolů do výše uvedených 3 skupin, ale přesto se stane, že plnění důležitých bezpečnostních požadavků je v některých sprintech vynecháno (Microsoft, 2012, str. 67).

3.1.2.3 DevSecOps

DevOps je kombinací slov Developement (vývoj) a Operations (provoz), která představuje základní myšlenku spolupráce mezi těmito na první pohled různými obory při vývoji softwaru[24]. DevSecOps[25] znamená integraci kontinuálních bezpečnostních principů, procesů a technologií do DevOps kultury a praxe – tedy místo, kde nalézají bezpečnost, provoz a vývoj společnou řeč (Tse, 2019, str. 8).

Konkrétní řešení nabízí společnost OWASP v podobě modelu SAMM – Software Assurance Maturity Model. Cílem modelu je poskytnout efektivní a měřitelný způsob, jak

[23] Tzv. Bucket Requirements.

[24] Přístup DevOps, poprvé zveřejněný během konference roku 2009, má své kořeny v agilním vývoji a důraz je zde kladen na princip upřednostnění jednotlivce a interakce nad procesy a nástroji. Dalšími cíli, jsou automatizace a Continuous Delivery (CD) – kontinuální dodání, viz Verona, 2018, str. 19.

[25] Kromě DevSecOps rozlišuje Tse ještě SecDevOps (aplikace kultury, postupů a pracovních postupů DevOps pro dosažení zabezpečení informací a správy dodržování předpisů) a DevOpsSec (aplikace principů a postupů zabezpečení informací na ochranu procesů, které využívají kulturu, postupy a pracovní postupy DevOps), blíže viz Tse, 2019, str. 7).

analyzovat a zlepšit cyklus bezpečného vývoje. Tohoto modelu využívá např. i společnost Dell za účelem určení prioritních komponent v bezpečném vývoji aplikací[26] (OWASP, 2020).

3.2 Zranitelnosti webových aplikací

3.2.1 Architektura webových aplikací

Klasická architektura webových aplikací je založena na klient-server modelu, přičemž komunikace může probíhat synchronně i asynchronně. Nalézáme zde tři úrovně[27] úloh:

1) Prezentace dat ve webovém prohlížeči
2) Aplikační logika na aplikačním serveru
3) Databanka v pozadí jako vrstva perzistence

Komunikace mezi jednotlivými vrstvami probíhá v případě prezentační a aplikační vrstvy prostřednictvím HTTP[28] nebo šifrovaně přes HTTPS[29]. Mezi aplikační a perzistentní vrstvou se částečně také používají proprietární přenosové protokoly. Jde o synchronní komunikaci, pokud je po aktivitě uživatele ve webovém prohlížeči dotaz přenesen na webový server, tam zpracován a odpověď zaslána zpět. Mezi odesláním dat a zobrazením výsledku musí uživatel čekat a nedochází v tomto čase k aktualizaci stránky. Za účelem snížení doby čekání uživatele a také kvůli potřebě individuálně měnit jednotlivé elementy stránky aniž by bylo nutné ji znovu načíst, byly vyvinuty asynchronní metody. Asynchronní JavaScript a XML (Ajax) představuje koncept pro asynchronní přenos dat mezi webovým prohlížečem a webserverem[30] (Kofler a kol., 2019, str. 732).

Z pohledu bezpečnosti je třeba vést v patrnosti řadu komponent: webový server, programovací jazyk a na něm postavené frameworky[31], vlastní kód, databázový server atd.

[26] Zdrojový kód modelu SAMM je k dispozici na https://github.com/OWASP/samm.
[27] V angličtině 3 Tiers architecture.
[28] HTTP je zkratka pro HyperText Transfer Protocol, tedy protokol, který definuje způsob a strukturu zpráv, které si klient (prohlížeč) a server vyměňují (Kurose, 2014, str. 97).
[29] HTTPS je zkratka pro HyperText Transfer Protocol Secure a představuje protokol, díky kterému je možné mezi klientem a serverem komunikovat bezpečně. Nutný je zde certifikát, který může být vytvořen např. u certfifikační autority (Kofler a kol., 2019, str. 656).
[30] Pro asynchronní přenos dat se etablovaly různé merody: REST (Representational StateTransfer), JSON (JavaScript Object Notation), SOAP (Simple Object Access Protocol), proprietární datové formáty.
[31] Framework, česky rámec, představuje jisté zjednodušení pro vývojáře. Např. při psaní webové aplikace vývojář nemusí psát každý řádek v PHP či HTML atp., ale framework poskytuje vývojáři hotové stavební bloky pro mnoho běžných úkolů (RiskSense, 2020).

Každý z těchto elementů představuje potenciální riziko. Zvláštní skupinou problémů byla spojena s kódem prohlížeče, jeho chybovost a obsah zranitelností a jeho kompatibilita s webovou aplikací, např. u Internetu exploreru[32] (Kofler a kol., 2019, str. 732).

Netcraft zveřejňuje pravidelně statistiky nejpoužívanějších webserverů. K říjnu 2020 zajišťoval Nginx provoz na 34% všech stránek, předehnal tak o 7% v minulosti dlouhodobě dominující Apache. Webservery Microsoftu jsou nyní na poklesu s 8% podílu na trhu (Netcraft, 2020).

Za účelem implementace frontendových funkcionalit a aplikační logiky je k dispozici řada webových frameworků. Výhodou frameworků, oproti jednotlivému vývoji, je, že jsou komunitou nebo firmami dále vyvíjeny a případné bezpečnostní chyby jsou jimi opravovány. Často používaným frameworky jsou ASP Active Server Pages, ASP.NET, Angular, Google Web Toolkit, JSF Java Server faces, JQuery, PHP, Struts (Kofler a kol., 2019, str. 733). Některé poskytují kompletní balíčky, které zahrnují bezpečnostní kontroly jako autentizace, autorizace, logování, validace, šifrování a správu relací. Příkladem populárních open source Frameworků pro programovací jazyk JAVA jsou: Spring Security, Apache Shiro, PicketLink, OACC (Object Access Control) Framework (Hsu, 2018, str. 98).

Vezmeme-li v úvahu jednotlivé jazyky, pak v programovacích jazycích na straně serveru je pro tvorbu webových aplikací nejvíce používáno PHP (78,8%), následované se značným odstupem ASP.NET (9,5%), Ruby (4,2%), Java (3,3%) atd. U jazyků na klientské straně vede JavaScript (96,9%), následovaný Flashem s 2,3% (W3Techs, 2020). K ukládání dat z dynamicky generovatelných obsahů slouží databáze. Využívají se: MySQL, popř. MariaDB, Oracle, Microsoft SQL-Server, MongoDB, Microsoft Access, IBM DB2, Informix (Kofler a kol., 2019, str. 733).

3.2.2 Zranitelnosti, zneužití a jejich evidence

Zranitelnost je slabina v bezpečnostních postupech, návrhu systému, implementaci, interních kontrolách atd., která může být využita k porušení bezpečnostní politiky systému. Jinými slovy jde o možnost zneužití systému nebo vystavení hrozbě (Slade, 2006, str. 194).

[32] Americký úřad pro vnitřní bezpečnost (US Homeland Security) např. doporučoval přestat užívat Internet explorer úplně, neboť útočníci mohli získat kontrolu nad zařízením díky zranitelnosti v prohlížeči (Dinita, 2020).

Asi nejznámější seznam kategorií zranitelností přináší **OWASP Top Ten**, který definuje 10 nejdůležitějších bezpečnostních rizik pro webové aplikace (OWASP, 2017). Tato rizika budou na dalších stránkách podrobněji popsána. Důležité je však zmínit i další iniciativy v této oblasti.

Common Waekness Enumeration (CWE) je komunitně vytvořený seznam softwarových a hardwarových slabin, které mají bezpečnostní důsledky. Slabiny v tomto pojetí představují různé chyby v kódu, návrhu, architektuře. Společnost Mitre připravuje seznam 25 nejnebezpečnějších softwarových slabin (CWE, 2020). Mitre také dává k dispozici přímo seznam konkrétních zranitelností – **Common Vulnerabilities and Exposures** (CVE, 2020).

Hodný zmínky je Společný systém hodnoceni zranitelností - **Common Vulnerability Scoring System** (CVSS), který poskytuje způsob, jak zachytit základní charakteristiky zranitelnosti a vytvořit číselné skóre, ve kterém se odráží její závažnost. Skóre pomáhá organizacím správně posoudit a stanovit priority v procesech řízení zranitelností (FIRST, 2020). Americký NIST vede **Národní databázi zranitelností** (NVD), kde zobrazuje, krom databáze zranitelností, také aktuálně nalezené zranitelnosti identifikované CVE-číslem včetně určení CVSS-závažnosti. Uložená data umožňují automatizaci správy zranitelností (NIST, 2020).

Krom registrací zranitelností systémů došlo k vedení seznamu způsobů, jak zranitelnost zneužít – příkladem je **Exploit Database**. Databáze „zneužití“ je udržována společností Offensive Security, která nabízí certifikace v oblasti informační bezpečnosti a také penetrační testy. Exploit database je neziskový projekt, který převzal a udržuje také GHDB – Google Hacking Data Base. Dnes obsahuje údaje, jak vyhledávat ve vyhledávačích a dalších úložištích (Bing, GitHub atd.) citlivé informace (Exploit-DB, 2020).

3.2.2.1 Nejčastější rizika podle OWASP Top Ten

Podíváme-li se blíže na vztah následujících rizik OWASP a CWE-slabin, lze vypozorovat, že rizika definovaná OWASP představují nadřazené kategorie CWE-slabin v abstraktnější podobě, do kterých jsou přiřazena již konkrétní CWE ve stromové

struktuře, která může být dále konkretizována[33]. Dále budeme vycházet z terminologie rizik OWASP za doplnění některých konkrétních CWE-slabin.

Injekce

Jde o chyby způsobené SQL, NoSQL[34], OS[35], HTML[36], CSS[37] a LDAP injekcí, které vzniknou, když jsou nedůvěryhodná data vložena do interpreteru jako součást dotazu. Útočníkem vložená data přimějí interpreter k provedení nechtěných příkazů nebo umožní přístup k datům, bez řádné autorizace (OWASP, 2017, str. 7).

V CWE top 25 nalezneme slabinu CWE-89, známou jako **SQL-injenction**[38], spadající do této kategorie. Moderní webové aplikace poskytují uživatelům k dispozici nejčastěji dynamické obsahy. Tzn., že některá data jsou zobrazena až na základě dotazu uživatele - po zadání určitých parametrů. Daná data jsou uložena v databázi a přistupuje se k nim prostřednictvím jazyka SQL (Structured Query Language).

Příkladem dotazu na data uživatele může být zjednodušeně: SELECT * FROM *uzivatele* WHERE *uzivatel*=’Karel’ AND *heslo*=’start123’, kde žádáme zobrazení uživatelových dat z tabulky *uzivatele* a výsledek omezujeme na konkrétního uživatele s daným heslem. U SQL-injenkce lze dotaz položit i bez znalosti konkrétního hesla, pokud to je aplikací umožněno. Dotaz může znít např. takto: SELECT * FROM *uzivatele* WHERE *uzivatel*=’Karel’ AND *heslo*=’xyz’ OR ‘1’ = ‘1’ . Je jedno tedy, jaké je skutečné heslo, neboť parametrem přidaným k dotazu ‘1’ = ‘1’ je dotaz vyhodnocen jako TRUE – pravdivý. Šlo o to, že původní SQL dotaz byl ukončen a doplněn vložením xyz’ OR ‘1’ = ‘1’ do vstupního pole pro heslo do přihlašovacího formuláře (Kofler a kol., 2019, str. 752).

Nefunkční autentizace

Funkce aplikace související s ověřováním a správou relací jsou často implementovány nesprávně, což umožňuje útočníkům prolomit hesla, klíče či tokeny, nebo

[33] Např. 1026 - Weaknesses in OWASP Top Ten (2017) je nadřazenou skupinou mimo jiné pro CWE CATEGORY: OWASP Top Ten 2017 Category A1 – Injection a v této skupině nalezneme např. CWE-564: SQL Injection: Hibernate.
[34] Injekce způsobená do databáze jiné než SQL. Dle OWASPu existuje přes 150 takových databází pro užití ve webových aplikacích. Příkladem je MongoDB (Marshall, 2018, str. 86).
[35] OS injekce umožňuje útočníkovi provádět libovolné příkazy operačního systému (OS) na serveru, na kterém je spuštěna aplikace a ohrozit aplikaci a její data (PortSwigger, 2020).
[36] Cílem útoku je vložit do stránky libovolný HTML kód.
[37] Injekce CSS kódu do webové stránky, který bude zobrazen v prohlížeči oběti.
[38] Více na: https://cwe.mitre.org/data/definitions/89.html.

zneužít další nedostatky k dočasnému či trvalému získání identity jiných uživatelů (OWASP, 2017, str. 7). Příkladem CWE-slabiny spadající do této kategorie je CWE-287 – nevhodná autentizace, kdy software neověřuje dostatečně, nebo vůbec, že daný uživatel má skutečně danou identitu. Do této skupiny patří mimo jiné následující slabiny: použití jednofaktorového ověření[39], slabé požadavky na heslo[40], použití „tvrdě nakódovaných" přihlašovacích údajů[41] atd.

Vyzrazení citlivých dat

Mnoho webových aplikací a API rozhraní nechrání správně citlivá data (např. osobní údaje, finanční data), která mohou útočníci ukrást, modifikovat, provést na jejich základě podvod, krádež identity nebo jiný zločin. Taková data mohou být dostupná díky nedostatečné ochraně, jako např. šifrování, a vyžadují ochranu při zasílání přes prohlížeč (OWASP, 2017, str. 7). Do této skupiny rizik patří např. CWE-slabiny neadekvátní síla šifrování[42], vystavení soukromých osobních údajů neautorizovanému aktérovi[43]atd.

XML external entities (XXE)

K útoku je zde využívána chyba v konfiguraci XML parseru aplikace, která může dále vést k dalším škodlivým akcím, jako vyzrazení obsahu chráněných souborů, způsobení exponenciálního užívání paměti, vedoucí k odepření služby – DOS (Denial of Service) útoku atd. XML zůstává ve starších API architekturách jako je SOAP (Simple Object Access Protocol), u nových se setkáváme spíše s JSON (Marshall, 2018, str. 119). CWE-slabinou patřící to této skupiny je např. nedostatečné omezení reference externí entity[44].

[39] Jde o větší riziko kompromitování účtu, než např. u dvoufaktorového ověření - CWE-308, blíže viz https://cwe.mitre.org/data/definitions/308.html.
[40] Produkt nepožaduje dostatečně silné heslo - CWE-521, blíže viz https://cwe.mitre.org/data/definitions/521.html.
[41]Jde o problematiku výslovně nakódovaných přihlašovacích údajů – hesel, kryptografických klíčů, které jsou např. užívány pro komunikaci s externími komponenty - CWE-798, blíže viz https://cwe.mitre.org/data/definitions/798.html.
[42] Software uchovává citlivá data, která jsou sice šifrována, ale ne dostatečným způsobem, jak by bylo pro situaci adekvátní – CWE-326, blíže viz https://cwe.mitre.org/data/definitions/326.html.
[43] Cizí aktéři nejsou výslovně autorizováni pro přístup k informacím nebo nemají výslovný souhlas osoby, o které jsou data sbírána – CWE-359, blíže viz https://cwe.mitre.org/data/definitions/359.html.
[44] CWE-611, blíže viz https://cwe.mitre.org/data/definitions/611.html.

Nefunkční kontrola přístupu

Restrikce, co mají povoleno ověření uživatelé, nejsou často správně vynucována. Útočníci mohou využít těchto chyb k získání přístupu k nepovoleným funkcionalitám či datům, přistoupit k jiným uživatelským účtům, nahlížet na citlivé soubory, měnit přístupová práva (OWASP, 2017, str. 7). Příkladem CWE-slabiny patřící do této kategorie je obejití autorizace prostřednictvím uživatelem ovládaného klíče[45].

Chybná konfigurace

Jedná se o nejběžnější pochybení jako následek nezabezpečených výchozích konfigurací, nekompletních ad hoc konfigurací, otevřeného cloudového úložiště, nesprávné konfigurace HTTP hlavičky, podrobné chybové zprávy obsahující citlivé informace. Nejenže musí každý operační systém, framework, knihovna a aplikace být bezpečně nakonfigurována, ale musí být i opravována (patchována) a upgradována včas (OWASP, 2017, str. 7). Příkladem CWE-slabin je např. kategorie chyb[46], kam patří chyba spočívající v ignorování výjimek a dalších chybových podmínek, které umožní útočníkovi nepozorovaně vyvolat neočekávané chování[47].

Cross-site Scripting (XSS)

Při skriptování napříč webem - XSS[48], které umožňuje útočníkovi spustit skripty v prohlížeči, není sice napaden webserver, ale lze získat cookies, ukrást uživateli relaci nebo např. přesměrovat uživatele na jinou škodlivou stránku. Během jednoduššího testu je možné zranitelnost XSS demonstrovat např. vepsáním do neošetřeného vyhledávacího políčka <script>alert(‘pozor XSS’)</script>. Pokud webová aplikace vložená data nijak nefiltruje, objeví se v prohlížeči JavaScriptové výstražné okno s upozorněním: *pozor XSS*. Reflektovaný XSS znamená, že kód je zadán přímo uživatelem, přičemž využitelnost je zde omezenější, protože uživatel by musel kliknout na připravený odkaz zaslaný emailem

[45] Útočníci se např. mohou podívat na místa, kde se načítají data specifická pro uživatele a určit, zda je klíč pro vyhledávanou položku externě ovladatelný. Klíč může mít podobu nezašifrované proměnné souboru cookies. CWE-639, blíže viz: https://cwe.mitre.org/data/definitions/639.html.
[46] CWE-388, blíže viz https://cwe.mitre.org/data/definitions/388.html.
[47] CWE-391: Unchecked Error Condition, blíže viz: https://cwe.mitre.org/data/definitions/391.html.
[48] Související CWE-slabina: CWE-79: nedostatečná neutralizace vstupu během generování stránky 'Cross-site Scripting'. Blíže viz: https://cwe.mitre.org/data/definitions/79.html.

nebo nastrčený na webové stránce. Při uloženém XSS (stored-XSS) je škodlivý kód uložen na stránce, kterou oběť musí nejprve navštívit.

Prostřednictvím XSS je také možné na dálku převzít ovládání infikovaného prohlížeče. Pohodlnou možnost k této aktivitě nabízí nástroj BEEF[49], který umožňuje také iniciovat lokální útoky vedoucí u zastaralých systémů plnému přístupu k systému oběti (Kofler a kol., 2019, str. 744).

Nezabezpečená deserializace

Útok může být proveden na každém systému, který serializuje a deserializuje data. Útočník změní aplikační logiku nebo se pokusí spustit vzdáleně kód, pokud jsou v aplikaci objekty, které mohou změnit chování nebo být spuštěné během nebo po deserializaci (Khawa, 2018, str. 142). V CWE-slabinách nalezneme chybu deserializace nedůvěryhodných dat[50]. Aplikace deserializuje nedůvěryhodná data, aniž by ověřila, že výsledná data budou validní.

Použití komponent se známými zranitelnostmi

Aplikace a API užívající komponenty (frameworky, knihovny, moduly,...) se známými zranitelnostmi mohou narušit obranu aplikací a způsobit různé útoky a dopady. Zneužití zranitelnosti může způsobit ztrátu dat nebo převzetí serveru útočníkem (OWASP, 2017, str. 7).

Nedostatečné logování a monitorování

Nedostatečná úprava logů a monitorování spolu s chybějící nebo neúčinnou integrací se systémem reakcí na incidenty[51] umožňuje útočníkům zaútočit na systém a v systému přetrvávat, dostat se k dalším částem systému a měnit, ničit či kopírovat data. Stude ukazují, že k detekování útoku došlo až po 200 dnech, typicky externím subjektem než interním procesem a monitoringem (OWASP, 2017, str. 7). Související CWE-slabinou je CWE-778-nedostatečné logování[52].

[49] BEEF je zkratkou pro Browser Exploitation Framework, blíže viz: https://beefproject.com/.
[50] Blíže viz: https://cwe.mitre.org/data/definitions/502.html.
[51] Tzv. Incident Response.
[52] Blíže viz https://cwe.mitre.org/data/definitions/778.html.

3.2.2.2 Zranitelnosti ve webových frameworcích

Za pozornost stojí statistická data porovnávající nalezené zranitelnosti v používaných programovacích jazycích a frameworcích. Zpráva RiskSense z března 2020 uvádí, že celkově se počet zranitelností v roce 2019 v porovnání s předešlými lety snížil, nicméně počet těch, které byly nebo mohly být zneužity exploitem[53] nebo malwarem[54] (byly tzv. weaponized - ozbrojeny), vzrostl dvojnásobně. Jenom samotné frameworky WordPress a Apache Struts zabíraly 55% všech "ozbrojených" zranitelností za posledních 10 let. Zatímco WordPress čelil škále problémů, mezi nimiž nejvýznamnější místo zabírá cross-site-scripting, u Struts dominovaly problémy s validací vstupů. Jazyky, na kterých jsou tyto frameworky postaveny PHP (u WordPress) a Java (u Struts) patří také k jazykům s největším počtem zranitelností, kde již existuje kód ke zneužití (RiskSense, 2020).

3.3 Metodologie testování webových aplikací

Existuje několik metodologií, vodítek a průvodců, jak se chopit bezpečnostního a penetračního testování[55]:

OWASP Testing Guide

Již několikrát citovaný průvodce zaměřený na webové aplikace. Představuje postup a detailní techniky testování v několika definovaných oblastech zejm. v podobě black box testů včetně následného reportingu.

ASVS OWASP

Standard ověření zabezpečení aplikace[56] představuje k říjnu 2020 aktualizovaný seznam požadavků a testů v oblasti aplikační bezpečnosti, které mohou být využity architekty, vývojáři, testery, bezpečnostními experty, prodejci nástrojů či spotřebiteli (OWASP, ASVS, 2020).

---

[53] Kód určený k nalezení/využití zranitelnosti, který může vést k dodání škodlivého kódu v další fázi útoku.
[54] Malware je škodlivý software – škodlivý program, který na počítači nebo jiném zařízení spustí nechtěnou funkci. Podle toho, jak se na infikovaném stroji rozšíří, mluvíme o virech, červech, trojských koních atd. (Kofler, 2019, str. 25).
[55] Existuje samozřejmě i řada standardů na „klasické" testování – tvorbu testovacího plánu, testovacích scénářů, techniky testování. O těch tato kapitola pojednávat nebude, popř. jen v menší míře za účelem vysvětlení základních pojmů.
[56] V originálním znění Appliacation Security Verification Standard.

OSSTMM[57]

Otevřená příručka metodiky testování bezpečnosti od ISECOM, kterou lze přizpůsobit jakémukoliv typu bezpečnostního auditu včetně penetračních testů, etického hackerství, posouzení zranitelností, aktivit červeného a modrého týmu[58] atd. Účelem dokumentu je poskytnout konkrétní popis testů provozní bezpečnosti zahrnující rozdílné provozní kanály: lidské, fyzické, bezdrátové, telekomunikační a síťové (Herzog, 2010).

Penetration Testing Guidance

Pokyny pro penetrační testování se zaměřují na komponenty pentestů, kvalifikaci pentestera, testovací metodologie – před testováním, během testování a aktivity po testování včetně pokynů pro reporting (PCI, 2017).

Technical Guide to Information Security Testing and Assessment

Technický průvodce testováním a hodnocením informační bezpečnosti poskytuje informace o technikách, plánování, exekuci bezpečnostních testů, definování opatření pro zmírnění dopadů nálezů (Scarfone a kol., 2008).

3.3.1 Příprava testů

Před počátkem testů je potřeba definovat, co chceme chránit[59], zmapovat okolí testovaného systému, k jakým dochází interakcím se systémem (také vstupy a výstupy do systému). Je potřeba vymezit rozsah a obsah testů[60]. Namístě je také zamyšlení nad tím, jaké informace chceme získat, jakým způsobem tak učiníme a jaké testy použijeme, např. black box testy atp. (Herzog, 2010, str. 33).

Důležitou roli zde hrají i tzv. pravidla zapojení (Rules of Engagement), a to zejm. u penetračních testů. Tato pravidla určují, co je v rámci testů dovoleno, jaké techniky se mohou použít (např. vyjmutí DOS-testů, používání automatizovaných scannerů), jaké domény či části aplikace jsou z testování vyloučeny, z jakých IP adres se smí testovat,

[57] OSSTMM je zkratkou pro The **O**pen **S**ource **S**ecurity **T**esting **M**ethodology **M**anual.
[58] Modrý tým je zodpovědný za ochranu systému. Červený tým testuje bezpečnost a ochranu dat testovaného systému – hledá a testuje přístupové body, hledá zranitelnosti, možnosti kompromitace serveru atd. (Kofler, 2019, str. 363).
[59] Jde o to definovat předmět testu – Asset.
[60] Tzv. scope.

jakým způsobem bude probíhat testovací proces a reportován výsledek (Marshall 2018, str. 28). Mimo jiné je doporučováno stanovit kritéria úspěchu testu[61], která umožní odpovídajícím způsobem nastavit limity hloubky penetračních testů. Příkladem může být kompromitování domény privilegovaným uživatelem (PCI, 2017, str. 17).

V rámci přípravy je nutné zajistit také nástroje pro testování. Existují linuxové distribuce obsahující předinstalované nástroje pro bezpečnostní testování. Jedná se např. o Kali Linux[62] a BlackArch[63]. Testovací prostředí pro operační systém Windows nabízí PentestBox[64] (Hsu, 2018, str. 198).

3.3.2 Provedení testů

Podle metodologie OWASP Testing Guide je test rozdělen do dvou fází: pasivního a aktivního modu. V **pasivním modu** se tester snaží porozumět aplikační logice a seznamuje (hraje si) s aplikací. Na konci této fáze by měl mít tester zmapované všechny přístupové body[65], např. HTTP hlavičky, parametry, cookies. V **aktivním modu** začíná tester testovat podle metodologie, která zahrnuje 11 kategorií s celkem 91 kontrolami. Z důvodu rozsáhlosti dokumentu bude popsáno 11 kategorií s pouze vybranými kontrolami (testy).

Zjišťování informací[66]

Tato kategorie navazuje na pasivní mod testování. Je doporučeno provést vyhledávání v prohlížeči pro únik informací. Přímé metody by souvisely s hledáním indexů a asociovaného obsahu mezipaměti. Nepřímé se vztahují na shromažďování citlivých informací o designu a konfiguraci prohledáváním fór, diskuzních skupin atd. Dalším krokem je zjistit verzi a typ webserveru, zjistit, jak server odpovídá na špatné dotazy (př. neexistující stránka). Dále je možné zkontrolovat meta-soubory webserveru (zaměřené na soubor robot.txt), zkontrolovat, zda se nachází na serveru další aplikace (nebo např. schované administrativní rozhraní), zda nejsou otevřené nestandartní porty a jestli v kódu nejsou ponechané zakomentované informace, které by mohly být zneužity,

[61] Tzv. Success Criteria.
[62] Viz https://www.wappalyzer.com/.
[63] Viz https://blackarch.org/.
[64] Viz https://pentestbox.org/.
[65] Tzv. Gates.
[66] Tzv. Information Gathering.

např. v html. V rámci identifikace vstupů je vhodné si projít metody používané při zasílání dotazů na server (hlavně POST a GET). Nástrojem pro testování je např: prohlížeč, httprint[67], Nmap[68], Nessus[69], OWASP: Zed Attack Proxy[70], Burp Suite[71], Wappalyzer[72] (OWASP, 2017, str. 50).

Testování konfigurace a nasazení

Zde jde o pochopení konfigurace serveru hostujícího webovou aplikaci. Testy se zaměřují na konfiguraci sítě/infrastruktury. Webová infrastruktura může obsahovat desítky aplikací a jediná zranitelnost může mít zásadní dopad. Konfigurace platformy, logování, kontrola záložních souborů pro obsah citlivých informací, zjištění rozhraní pro administrátory, detailní test HTTP metod a také přítomnost HSTS[73] hlavičky je předmětem této skupiny testů (OWASP, 2017, str. 64).

Testování řízení identit

V běžných moderních systémech jsou definovány rozdílné role pro správu uživatelů a definování přístupu k informacím a prostředkům. Elementární je alespoň rozdělení na 2 skupiny: správce, který má privilegovaný přístup k citlivým informacím a funkcím, a běžné uživatele přistupující k běžným funkcím. Tvrdá autorizace v podobě složité struktury rolí nemusí být jediným řešením, jak spravovat přístup do systémů, neboť v důvěryhodnějších prostředích lze využít měkčích ovládacích prvků, jako je definování různých toků (flow) v aplikacích a auditní logování. Definování několika velmi širokých rolí může být ale stejně nevhodné jako velké množství příliš úzce specifikovaných rolí.

Cílem testu je validovat definované role zmapováním rolí a prozkoumáním souvisejících oprávnění. Testovacím objektem zde bude také způsob registrace uživatelů, způsob přidělování oprávnění uvnitř aplikace, hádání uživatelských účtů a reakce systému na tuto aktivitu (OWASP, 2017, str. 68).

---

[67] Viz https://net-square.com/httprint.html.
[68] Viz https://nmap.org/.
[69] Viz https://www.tenable.com/products/nessus.
[70] Tzv. ZAP, viz https://www.zaproxy.org/.
[71] Viz https://portswigger.net/burp.
[72] Viz https://www.wappalyzer.com/.
[73] HSTS je zkratkou pro HTTP Strict Transport Security a představuje mechanismus, kterým lze zabránit např. útokům tzv, Man-in-the-Middle (tzv. muž uprostřed útoku). Díky HSTS je dána webserverem prohlížeči instrukce, že stránka má komunikovat jen zašifrovaně přes HTTPS (Kofler, 2019, str. 783).

Testování autentizace

Autentizace koncového bodu je proces, při kterém jeden subjekt prokazuje svou totožnost jinému subjektu přes počítačovou síť (Kurose a kol. 2014). Testování se soustřeďuje na ověření, zda jsou přihlašovací údaje přenesené bezpečným, zašifrovaným kanálem z prohlížeče na server, jestli aplikace užívá HTTPS, zda je uživatel ověřen a zobrazí se mu požadovaná stránka. Další testy se zaměřují na to, zda uživatel/é nepoužívají defaultní – původně přidělené přihlašovací údaje (OWASP, 2017, str. 72).

Dále se testy zaměřují na zamykací mechanismy po neúspěšných přihlášeních a potenciál na brute-force útok, který spočívá v tom, že tzv. hrubou silou hádáme heslo uživatele. Možnosti takového útoku se odvodí od počtu možných hesel (možno odvodit podle délky hesla a povolených znaků) a délce trvání testu jednoho hesla. S komplexností hesla stoupá čas věnovaný prolomení hesla. Časově méně náročný je slovníkový útok, kde lze využít skutečnosti, že některé pojmy jsou obecně oblíbené jako hesla a s takovým seznamem hesel se snaží útočník automatizovaně prolomit přístup do aplikace (Kofler a kol., 2019).

V rámci testů autentizace se testy zaměřují i na možnost obejít autentizační mechanismus a např. přímo zobrazit následnou stránku přeskočením přihlašovací stránky, dále, jestli nedochází k ukládání hesla např. v prohlížeči, zda není slabá politika hesel, zda bezpečnostní otázka/odpověď není slabá a nebo není slabý mechanismus resetu hesla (OWASP, 2017, str. 79).

Testování autorizace

Po autentizaci je v aplikaci určeno, který uživatel bude mít přístup na které objekty (data, dokumenty, položky navigace atd.). Toto přiřazení a ověření přístupových práv je nazýváno autorizací (Kofler a kol. 2019, str. 734). Probíhají testy, zda lze obejít autorizační schéma, jestli se nedají přidělit privilegovanější práva a ověřuje se možnost přistoupit k souborům, které byly zamýšleny jako nepřístupné (OWASP, 2017, str. 87).

Testování správy relací

Jednou ze součástí webové aplikace je mechanismus, kterým se řídí a udržuje stav interakce uživatele s aplikací. Správa relací je pokryta od autentizace až po odhlášení. Protokol HTTP je bezestavový protokol, což znamená, že webserver odpovídá na dotaz ze strany klienta, aniž by se navzájem propojovaly. I v jednoduché aplikaci je potřeba, aby

bylo spojeno více požadavků uživatele v jedné relaci. Nejpopulárnější webové prostředí postavené na ASP nebo PHP poskytují uživatelům vestavěné zpracování relací, kdy je přiděleno session ID nebo cookie. Cílem testů v této kategorii je ověřit schéma relací, atributy cookies, fixace relace, vystavení proměnných relace a funkcionalitu odhlášení (OWASP, 2017, str. 94).

Lze také provést útok CSRF (Cross-site Request Forgery), ve kterém útočník využije uživatelovy relace. Útok vyžaduje sociální inženýrství, při kterém je uživatel např. vylákán na infikovanou stránku a díky aktivní relaci uživatele a znalosti jeho session ID či cookies může útočník provést aktivity za uživatele – např. přidat článek, do blogu, změnit E-mail, provést platbu atd. (Khawaja, 2018, str. 131).

Testování validace vstupů

Chyba spočívající v nedostatečné validaci na straně klienta nebo prostředí je nejběžnější bezpečnostní slabinou, která může vést ke XSS, nebo SQL, NoSQL, LDAP, XML, XPath, injekci, přetečení paměti atd. Namístě je tedy provést testy na tyto zranitelnosti – XSS a injekce jsou popsány již výše. Lze také ověřit, zda server nepřijímá jiné metody než GET a POST (OWASP, 2017, str. 155).

Ošetření chyb

Během testování narazí tester na chybové hlášky generované aplikací nebo serverem. Chyby je možné vyvolat manuálně či speciálními nástroji. Chyby jsou významné, protože prozrazují často informace o databázích, defektech, webserverech a technických komponentech. Řada chyb, které hlásí databáze, mohou být dále užité pro SQL injekci. Pro testování lze využít nástrojů ErrorMint, ZAP OWASP (OWASP, 2017, str. 160).

Kryptografie

Jde o testování ochrany přenášených dat: testování slabin SSL-TLS, ochrany transportní vrstvy, validity certifikátů, šifrování kanálů. Do této skupiny testů patří i útok tzv. „polstrovaných věštců“ (Padding Oracle Attack). Jde o funkci, která dešifruje šifrovaná data poskytovaná klientem (např. stav interní relace uložené na straně klienta) a dojde k úniku stavu platnosti polstrování po dešifrování. Existence polstrovaného věštce umožňuje útočníkovi dešifrovat zašifrovaná data a šifrovat libovolná data bez znalosti

klíče použitého pro tyto kryptografické operace. Útoku se využívá většinou v blokových šifrách[74]. Nástrojem pro testování jsou: PadBuster, Python-PaddingOracle, Poracle, Padding Oracle Exploitation Tool – POET. (OWASP, 2017, str. 175).

Testování obchodní (business) logiky

Tento druh testování vyžaduje trochu jiný způsob přemýšlení testera než u ostatních bezpečnostních testů, podobá se funkčnímu testování a primárně se neautomatizuje. Testy se soustřeďují např. na validaci dat – jen logicky validní data by měla být vložena na frontendech, stejně jako na straně serveru (možno testovat prostřednictvím OWASP Zed Attack Proxy). Dále se testuje možnost podvrhnout požadavky při cestě na backend. Další test obnáší zjištění specifik v trvání procesů (např. neúspěšný login a generování chybové hlášky je rychlejší než načtení stránky po úspěšné autentizaci), testování limitů, jak často může být funkce užita (např. lze slevový kupón uplatnit ještě jednou), obejití pracovních toků (work flows), možnost nahrát neočekávané či škodlivé soubory (OWASP, 2017, str. 190).

Testování na straně klienta

Jedná se o testy zahrnující spuštění kódu na straně klienta, typicky v prohlížeči nebo prostřednictvím pluginu prohlížeče. Kromě výše popsaného XSS a injekcí lze zmínit při testování útok zaměřený na manipulaci se zdroji na straně klienta (Client Side Resource Manipulation), kdy aplikace přijme uživatelem řízený vstup, který specifikuje cestu zdroje (např. zdroj iframu) s tím, že dopad závisí typu elementu, jehož URL je kontrolovaná útočníkem (OWASP, 2017, str. 202).

Dalším testem v této kategorii je tzv. *clickjacking*, ve kterém útočník schová škodlivý link pod legitimní elementy stránky a uživatel je kliknutím naveden na tuto stránku (Marshall, 2018, str. 202).

3.3.3 Reporting a doporučení

Na závěr testování vyhotovují testeři zprávu, ve které je strukturovaně komunikováno, co a jak bylo testováno, jaká metodologie a nástroje byly použity, jaké

[74] Bloková šifra je jednou ze symetrických šifrovacích technik, kdy se zpráva, která se má zašifrovat, zpracovává po blocích k-bitů. Pokud se k=64, pak je zpráva rozdělena do 64bitových bloků a každý blok se šifruje nezávisle. Při šifrování se mapuje jedna ku jedné – k-bitový blok prostého textu se mapuje na k-bitový blok šifrovaného textu (Kurose, 2014, str. 523).

byly limity testů. Poté následuje popis výsledku testů. Nalezené zranitelnosti se ohodnotí podle závažnosti a způsob udělení tohoto označení by měl být vysvětlen (např. užití standardu CVSS). Vhodné je nález obohatit referencí na existující slabiny, popř. zranitelnosti podle uznávaných standardů, př. CVE, CWE identifikátor (PCI, 2017, str. 24).

Metodologie OSSTMM doporučuje k reportování STAR zprávu (Security Test Audit Report), která přesně kalkuluje plochu útoku testované aplikace. Pokud zpráva obsahuje návrh řešení a doporučení, jak zranitelnost odhalenou testováním odstranit, mělo by se jednat o doporučení validní a praktické (Herzog, 2010).

4 Vlastní práce

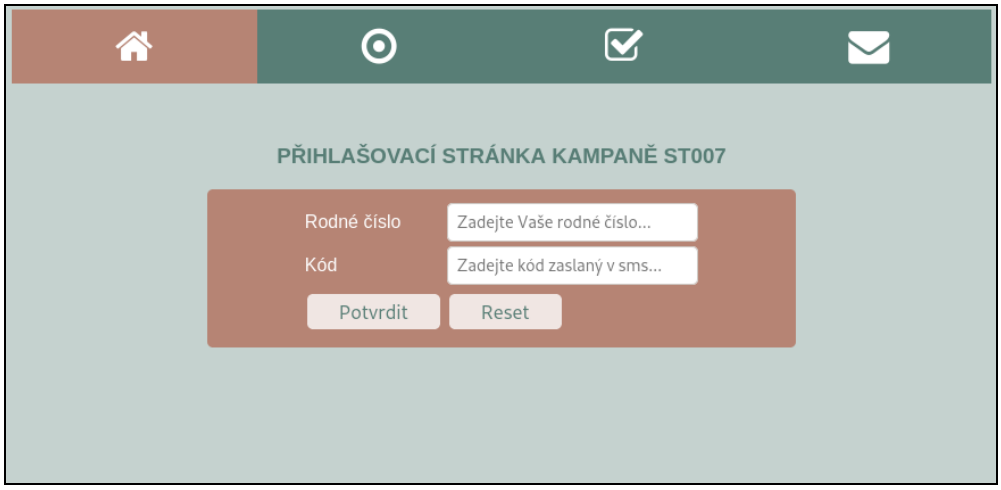
V praktické části půjde o testování webové aplikace vytvořené autorkou práce. Aby mohlo být ukázáno širší spektrum testů nepůjde o klasickou testovací situaci – autorka kódu je zde zároveň testerem ve všech fázích vývoje. Byť je autorce znám zdrojový kód, budou provedeny i tzv. black box testy.

4.1 Testovaná aplikace

4.1.1 Funkční specifikace aplikace

Předmětem bezpečnostních testů je webová aplikace nazvaná Kampaň. Cílem kampaně je získat vyjádření klienta – uživatele, se zpracováním osobních údajů a v případě souhlasu získat i E-mail klienta pro další komunikaci.

Aplikace se skládá ze tří po sobě jdoucích stránek a stránky s kontaktním formulářem. Uživatel se přihlásí prostřednictvím přihlašovacího formuláře. Jako přihlašovací údaje slouží rodné číslo a heslo představuje v SMS zaslaný číselný kód. Zadané údaje klient potvrdí tlačítkem „Potvrdit“. V případě stisknutí tlačítka „Reset“ dojde ke smazání údajů vyplněných v polích pro rodné číslo a kód.

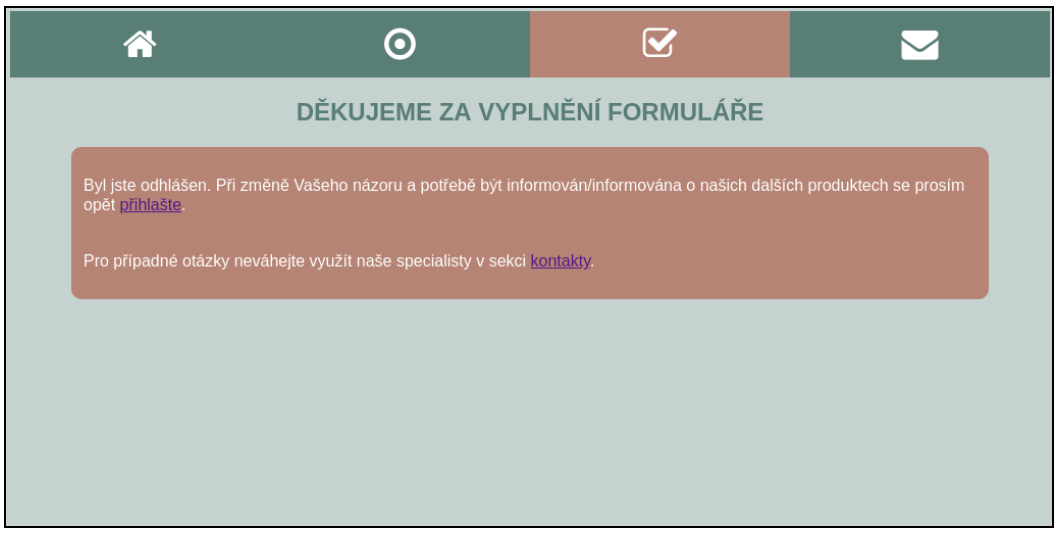


Obrázek 2: Přihlašovací stránka testované aplikace

Po přihlášení následuje stránka, kde se klient rozhodne, zda souhlasí se zpracováním údajů a možností být kontaktován pro marketingové účely. Klient může zanechat pro kontakt i E-mail – v případě souhlasu je to dokonce nutnou podmínkou pro pokračování na potvrzující stránku. V případě kliknutí na tlačítko „Zrušit“ se zobrazí přihlašovací stránka.

Obrázek 3: Stránka s výběrem rozhodnutí

Po potvrzení výběru vhodné varianty následuje potvrzující stránka s informacemi, co činit v případě změny rozhodnutí. Zároveň může klient zobrazit kontaktní formulář.



Obrázek 4: Potvrzující stránka

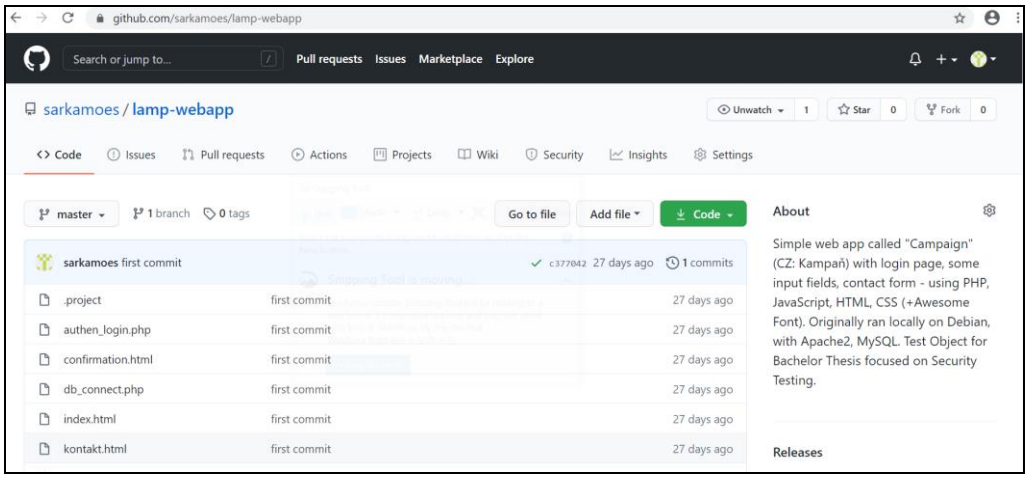
Kontaktní formulář slouží pro další možný kontakt klienta a obchodního zástupce. Tato stránka je přístupná uživatelům i bez přihlášení.

Obrázek 5: Kontaktní formulář

4.1.2 Technická specifikace

Aplikace byla napsána v platformě LAMP – v operačním systému Linux, za použití webserveru Apache/2.4.46 (Debian) v programovacím jazyce PHP. Tento programovací jazyk byl vybrán úmyslně vzhledem k tomu, že je stále v převažující většině používán při vývoji webových aplikací (viz statistická data výše).

Datová část je založená na databázi MariaDB verzi 10.3.24. Frontendová část je vytvořena v HTML, CSS a JavaScript. Aplikace byla vyvíjena lokálně, ale zdrojový kód je k dispozici na GitHub[75] a také v příloze č. 2 této práce.



Obrázek 6: Projekt na GitHub s uloženým zdrojovým kódem

[75] Jedná se o repozitář LAMP-webapp na https://github.com/sarkamoes/lamp-webapp.

4.2 Příprava testů a nástrojů

Kdybychom vzali v úvahu fáze bezpečnostního vývoje, budou autorkou práce provedeny vybrané testy ve fázi implementace softwaru a před nasazením softwaru do produkce. Ve fázi implementace půjde o statickou analýzu kódu za pomoci nástroje SonarQube. SonarQube je automatizovaný nástroj pro kontrolu kódu, který vyhledává defekty, náchylnosti k chybám a také bezpečnostní zranitelnosti (SonarQube, 2020). V této práci se zaměříme pouze na vybranou část analýzy kódu, která se týká identifikovaných zranitelností. SonarQube poskytuje zdarma možnost analyzovat z hlediska bezpečnosti repozitáře zveřejněné na GitHubu[76]. Výstupy analýzy jsou dostupné v cloudové službě SonarCloud.[77] Bude se jednat o tzv. white box test.

Ve fázi verifikace vývoje softwaru budou použity vybrané testy podle metodologie OWASP Testing Guide. Základní testovací prostředí bude v distribuci OS Linux – Kali Linux. Tato distribuce je zaměřená na penetrační testy a bezpečnostní audit a obsahuje několik set nástrojů určené pro řadu úkonů: penetrační testy, bezpečnostní průzkum, forenzní analýzu (Kali, 2020). Aplikace bude testována v prohlížeči Firefox, verze 78.3.Oesr (64-bit). Cílem testů je zjistit zranitelnosti ve připravené webové aplikaci. Testování proběhne zejm. jako black box test (popř. grey box test) a bude provedeno jen lokálně, neboť aplikace není publikovaná do internetu. Pro účely testu jsou dány k dispozici dva testovací účty pro přihlášení do aplikace: jedná se v obou případech o rodné číslo a šestimístné číslo jako kód (heslo).

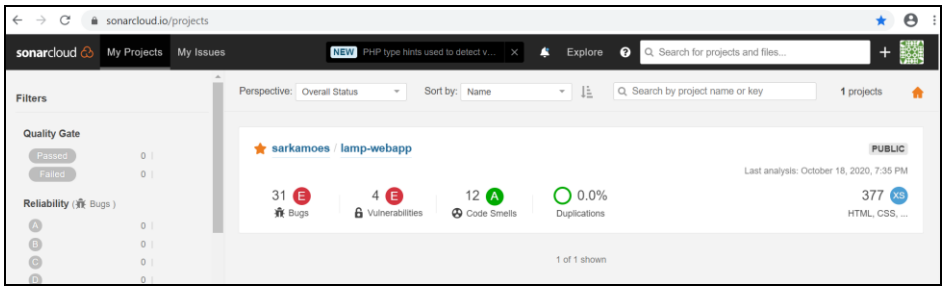
Nálezy v obou fázích vývojového cyklu budou vzestupně číslovány od N1 až po Nn. Závažnost nálezů bude hodnocena podle metodologie CVSS, tedy podle 5 stupňové škály: žádná závažnost (None), nízká (Low), střední (Medium), vysoká (High), kritická (Critical).

4.3 Provedení testu a výsledky

4.3.1 Testování bezpečnosti zdrojového kódu

Jak již bylo popsáno výše, byl k statické analýze kódu využit nástroj SonarQube.

[76] U soukromých, neveřejných, repozitářů na GitHubu je analýza sonarem zpoplatněna.
[77] Viz https://sonarcloud.io.



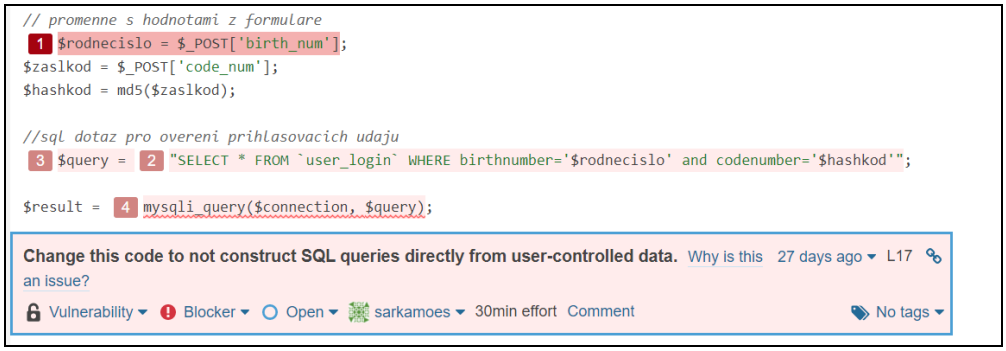
Obrázek 7: Náhled na výsledek analýzy SonarQube

Během skenu SonarQube bylo nalezeno 31 chyb, které se týkaly html kódu a také CSS (např. špatný hexadecimální kód barvy). Došlo k identifikaci 12 míst v kódu, které sice nejsou defekty, ale zaslouží si pozornost: např. nadbytečný kód, který byl tzv. „zakomentován“ během vývoje, ale nebyl posléze odstraněn.

Detailněji se zaměříme na 4 identifikované bezpečnostní zranitelnosti a 1 bezpečnostní hotspot[78], které lze shrnout do 4 nálezů:

N1: SQL dotazy založené na uživatelem kontrolovaných datech (závažnost kritická)

Na několika místech zdrojového kódu dochází ke konstrukci SQL dotazu za přímého použití hodnot, které zadá uživatel. Např. při vkládání hodnot do vstupních polí v přihlašovacím formuláři. Uživatelem poskytnuté údaje by měly být považovány jako nedůvěryhodné. Útočníkovi je umožněno vkládat hodnoty, které mohou měnit počáteční význam samotného dotazu. Takto upravené úspěšné dotazy mohou číst, upravovat nebo mazat citlivé informace databáze.



Obrázek 8: Náhled na nalezenou zranitelnost v SonarQube

[78] Security Hotspot je místo v kódu, které není jednoznačně zranitelné a o jeho zranitelnosti musí být na základě posouzení rozhodnuto.

N2: Chybějící ochrana databáze prostřednictvím hesla (závažnost kritická)

V místě kódu, kde je popsáno připojení k databázi, je uveden uživatel a namísto hesla je zde prázdný řetězec, což na první pohled indikuje, že se jedná o heslem nechráněnou databázi. Databáze by vždy měly být chráněny heslem.

N3: Nevhodné reflektování uživatelem kontrolovaných dat (závažnost kritická)

Jedná se vypsání chybové hlášky, která obsahuje přímo uživatelem zadaná data. Toto může umožnit uživateli injektovat škodlivý kód a spustit ho v prohlížeči. Důsledkem může být přístup či změna citlivých informací (SonarCloud, 2020).

N4: Slabý hašovací algoritmus (závažnost vysoká)

Heslo pro přístup do testované aplikace není v databázi uloženo v prostém textu, ale je zahašováno[79] prostřednictvím hašovací funkce MD5[80]. Takový hašovací algoritmus není již aktuálně považovaný za bezpečný, neboť je velmi jednoduché vytvořit tzv. hašovací kolize. Tzn., že stačí malé výpočetní úsilí pro nalezení dvou nebo více vstupů, které produkují stejný haš (SonarCloud, 2020). MD5 kolize se dají vypočítat aktuálně v řádech několika málo hodin a na internetu existuje mnoho stránek s kalkulátory, které umožňují vypočítat z haše chtěný řetězec. Kromě MD5 se z bezpečnostních důvodů nedoporučuje používat ani MD2, MD4 či SHA-1 (Kofler a kol., 2019, str. 242).

4.3.2 Bezpečností testy ve fázi verifikace

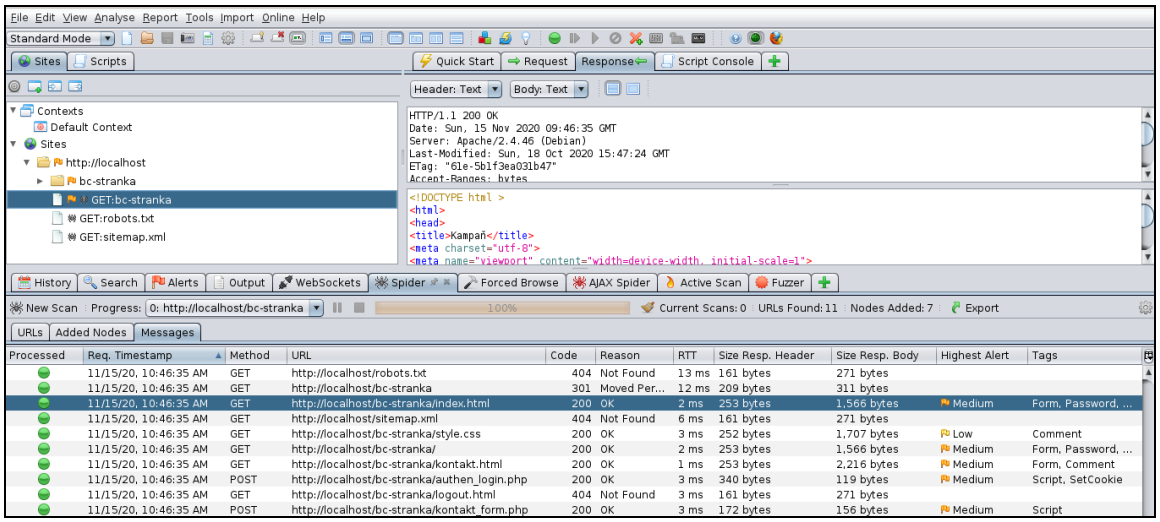
Zjišťování informací a konfigurace

Řada informací lze zjistit přímo z webového prohlížeče. Např.: vepsáním libovolné hodnoty do přihlašovacího formuláře a potvrzením tlačítka přihlásit lze přímo v prohlížeči v rozhraní pro vývojáře (Developer Tools Interface) v sekci síť nalézt v hlavičce odpovědi požadavku (metoda POST) informaci o serveru – Apache/2.4.46 (Debian). Takto byla manuálně zkoumána odpověď serveru na dotazy i na neexistující stránky. K tomuto účelu

[79] Při hašování je použita tzv. hašovací funkce, která podle vstupu vypočítává řetězec s pevnou velikostí – tzv. haš/hash (Kurose, 2014, str. 531).

[80] MD5 je velmi rozšířeným hašovacím algoritmem od Rona Rivesta, který vypočítává 128bitovou hašovací hodnotu. Proces sestává ze 4 kroků: přidání výplně, aby zpráva měla požadovanou délku; přidání 64bitové reprezentace délky zprávy před výplní; inicializace akumulátoru a závěrečného opakovacího kroku, kdy se zpráva ve 4 kolech rozdělí do bloku složených z 16 slov (Kurose , 2014, str. 531).

byl použit také automatizovaný nástroj Spider v OWASP ZAP, který umožňuje během skenu identifikovat další linky a odkazy na nové zdroje, parsovat odpovědi a zjistit nedostatky v nastavení serveru. V prohlížeči Firefox byl využit plugin FoxyProxy standard ve verzi 7.5.1, který umožňuje prakticky přepínat nastavení proxy, bez opakovaného a zdlouhavého hledání v nastavení prohlížeče. Také byl využit plugin Wappalyzer, pro zmapování použitých technologií. Tento nástroj označil verzi serveru, ale také např. použití Font Awesome, který byl využit pro položky navigace na stránkách aplikace.



Obrázek 9: Uživatelské rozhraní OWASP ZAP a Spider

Nástroj Spider se řadí do skupiny tzv. pasivního skenu, kdy nedochází ke změně dotazů na serveru, na rozdíl od aktivního skenu, při kterém je využíváno známých útoků a zranitelností (Zaproxy, 2020). Během tohoto testu bylo identifikováno 5 nálezů[81].

N5: Nenastavení X-Frame volby v hlavičce odpovědi (závažnost střední)

V HTTP odpovědích serveru není nastavena X-Frame volba[82], která by umožnila chránit aplikaci před potenciálním clickjacking útokem. Chyba je zařazena v seznamu slabin pod číslem CWE-16, v kategorii konfigurace (Zaproxy, CWE, 2020).

N6: Absence tokenu proti CSRF útoku (závažnost nízká)

Útok CSRF byl popsán již výše. Při odesílání dat v požadavcích směřujících na server prostřednictvím formuláře nebyl nalezen token, který by znemožňoval

[81] Další 2 nálezy, které nástroj Spider označil v závažnosti ‚Information' (pro informaci) nebudou na dalších stránkách dále analyzovány.
[82] V angličtině X-Frame Options Header.

potenciální CSRF útok. Chyba je zařazena v seznamu slabin pod číslem CWE-352, popisuje detailně CSRF a spadá do skupiny chyb spočívajících v nedostatečném ověření autenticity přijímaných dat (Zaproxy, CWE, 2020).

N7: Cookie bez HttpOnly flagu (závažnost nízká)

Aplikací užívané cookies bylo nastaveno bez HttpOnly flagu („praporku“), což znamená že na cookies lze přistupovat např. pomocí JavaScriptu. Pokud může být na stránce spuštěn závadný kód, cookies bude přístupná a může být přenesena na jinou stránku. V případě relačních cookies[83] může hrozit i únos relace[84] (Zapproxy, 2020). Při přihlášení uživatele je webserverem generováno ID relace (Session ID), které je zasláno v rámci každého požadavku na server. Pokud je přenášená cizí (ukradená) ID relace, znamená to, že může útočník v relaci napadeného uživatele číst a měnit data a došlo k únosu relace (Kofler a kol., 2019, str. 737). Tato slabina se řadí do již zmíněné kategorie slabin konfigurace CWE-16 (CWE, 2020).

N8: Cookie bez atributu SameSite (závažnost nízká)

Cookie bylo nastaveno bez atributu SameSite, což znamená, že může být zaslána jako výsledek požadavku napříč různými stránkami (cross site request). Nastavení tohoto atributu zabraňuje CSRF útoku a jiným útokům (Zaproxy, 2020).

N9: Chybějící nastavení X-Content-Type-Options (závažnost nízká)

Absence takového nastavení umožňuje ve starších verzích prohlížeče Internet Explorer a Chrome provést tzv. MIME[85]-sniffing. To znamená, že prohlížeče hledají správný MIME typ a pokud není nastaven, hádají ho. Existuje podezření, že některé MIME typy reprezentují spustitelný obsah, který může být škodlivý (MDN, 2020).

Testování řízení identit

Již ze zadání funkční specifikace a povahy testovacích účtů je patrné, že v aplikaci nedochází k žádnému systému rozdělení uživatelů podle rolí. Každý uživatel má stejná

[83] V angličtině Session Cookie.
[84] V angličtině session hijacking.
[85] MIME je zkratkou pro Multipurpose Internet Mail Extension. MIME typ je pak standard, který indikuje např. povahu a formát dokumentů, souborů. Prohlížeče používají MIME typ, aby určily, jak procesovat URL a je důležité, aby webservery posílaly správný typ v odpovědní hlavičce, položce Content-Type.

práva a možnost vyjádřit souhlas/nesouhlas se zpracováním osobních údajů a zpracováním pro marketingové účely – funkce je pro všechny přihlášené uživatele stejná. Registrace či vytváření uživatelů není v aplikaci zavedena, není tedy ani předmětem testu.

Autentizace a kryptografie

Při zadání neplatných přihlašovacích údajů je uživateli oznámeno prostřednictvím JavaScriptového okna, že přihlašovací údaje jsou neplatné. V případě zadání správných přihlašovacích údajů se zobrazí uvítací stránka aplikace s dalšími instrukcemi pro uživatele ohledně rozhodnutí o zpracování osobních údajů. V aplikaci se nenachází způsob, jak heslo resetovat. Toto je však vlastnost, nikoliv chyba a k potřebným resetům hesla nedochází v tomto grafickém rozhraní testované webové aplikace.

Byla testována i zranitelnost aplikace na defaultní hesla a názvy testovacích účtů (uživatel: 1234567890, admin; heslo: test, heslo123, 123456 atd.), ale bez nálezu. Byl testovaný způsob zamykacích mechanismů a zranitelnost na tzv. útoky hrubou silou, přičemž byly identifikovány následující nálezy:

N10: Absence zamykacích mechanismů při přihlášení (závažnost vysoká)

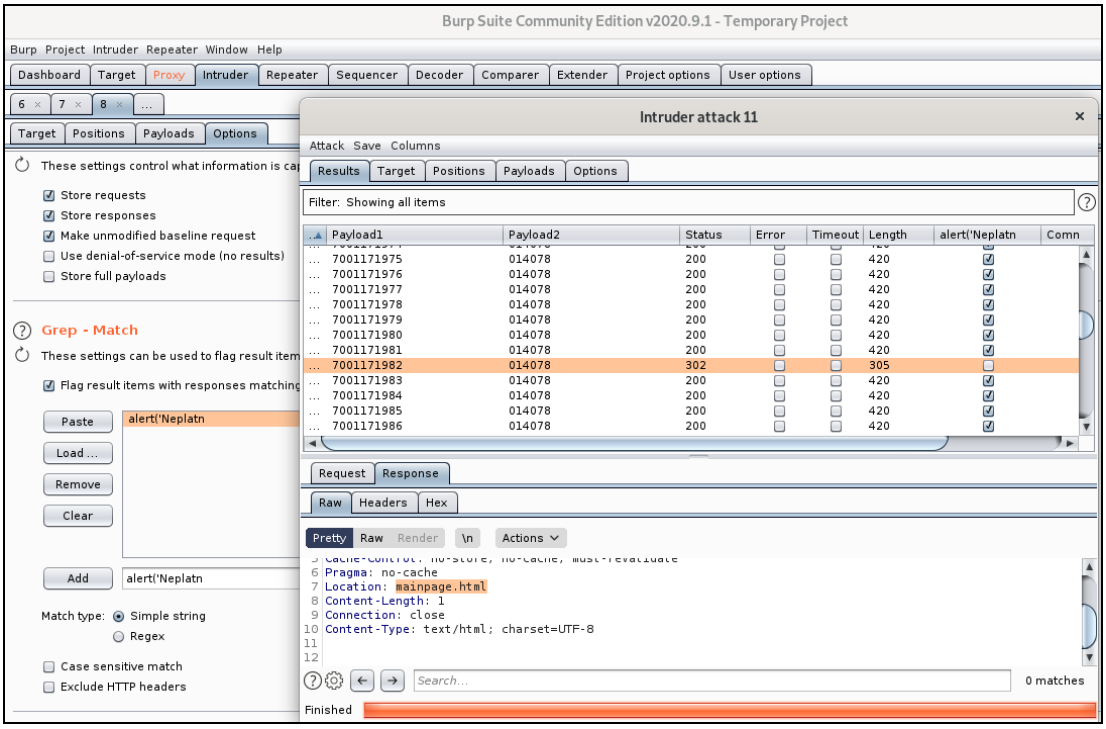
S poskytnutými testovacími účty bylo testováno, jak aplikace reaguje na opakované zadání špatného hesla: po třech zadáních špatného hesla bylo zadáno heslo správné, které umožnilo přihlášení do aplikace (počet neúspěšných pokusů předcházejících přihlášení úspěšnému se postupně zvyšoval). Aplikace reaguje způsobem, který značí, že neexistuje mechanismus pro zamknutí hesla po určitém počtu špatných pokusů či zpomalení odpovědi serveru na opakovaný vyšší počet neúspěšných požadavků.

N11: Aplikace není chráněna před automatizovanými útoky (závažnost kritická)

Ze specifikace a dostupných uživatelských účtů je patrné, že přihlašovací jméno má podobu celého 9 až 10-místného čísla, neboť jde o rodné číslo. Přístupovým kódem je pak pravděpodobně 6-místné číslo. Byl proveden automatizovaný útok hrubou silou (brute force attack) za pomoci nástroje Burp Suite.

V nástroji Burp Suite v sekci Proxy byl „odchycen“ požadavek směrovaný na server, který byl vytvořen zadáním libovolných přihlašovacích údajů a jejich potvrzením v testované aplikaci. Tento požadavek byl zaslán do sekce Intruder Attack (útok narušitele), byl definován typ útoku (Cluster Bomb), byla vymezena místa

v požadavku, kam budou hrubou silou postupně dosazovány hádané hodnoty. Je možné definovat, jaké hodnoty budou v rámci automatizovaných pokusů do přihlašovacích polí vkládány (sekce Payload). Omezeny byly na číselné hodnoty o tipovaném, výše popsaném, rozsahu. Vhodné je v sekci Možnosti také uvést, jakou hodnotu očekáváme v odpovědi na požadavek v případě špatného přihlášení (např. že se v hlavičce odpovědi v našem případě objeví alert JavaScriptu se zprávou o neplaných údajích), aby šlo neplatné přihlášení odlišit snadno od platného. Útokem byl nakonec přístup do aplikace prolomen a byly získány platné přihlašovací údaje[86].



Obrázek 10: Burp Suite se zobrazením řádky, kde došlo k prolomení hesla

N12: Absence protokolu HTTPS (závažnost kritická)

Protokol HTTPS umožňuje zašifrovanou komunikaci díky použití TLS/SSL. Aktuálně není HTTPS na serveru nastaven, což představuje závažnou bezpečnostní slabinu.

[86] Pro účely demonstrace zranitelnosti v tomto testu byl rozsah čísel rodného čísla a hesla zmenšen, než v případě rozsahu reálného útoku, který by v našem případě trval značnou dobu, vzhledem k autorce dostupné licenci Burp Suite. Nástroj v komunitní verzi dostupné zdarma umožňuje zpočátku automaticky procesovat cca 1 přihlášení za sekundu, po zpracování 1000 požadavků však rychlost značně klesne až na zpracování 1 požadavku za 1 minutu. Toto zřejmě má motivovat uživatele ke koupi pokročilejší licence.

Testování autorizace

Bylo provedeno několik testů, ve kterých bylo prověřováno, zda může uživatel přistoupit k informacím, ke kterým by neměl mít přístup a jestli může načíst stránku, kterou by načíst neměl nebo měnit na takto načtené stránce údaje. Šlo např. o scénář, ve kterém se uživatel přihlásí do aplikace a na stránce mainpage.html (místě, kde rozhoduje o souhlasu se zpracováním osobních údajů) stiskne tlačítko zrušit. Načetla se opět hlavní stránka. Poté byla stránka mainpage.html zadána přímo do URL-pole prohlížeče bez předchozího nového přihlášení uživatele a „jiný“ uživatel zkusí změnit hodnotu uloženou v databázi[87] - změní v uživatelském prostředí hodnotu z NE na ANO a vyplní E-mail a své rozhodnutí potvrdí. Po tomto kroku se načte stránka s potvrzením a díky za vyplnění formuláře. Pozitivní je skutečnost, že původní hodnoty v databázi nebyly touto neoprávněnou akcí změněny, tudíž data zadána neoprávněným uživatelem nebyla uložena. Uživatel, který se však pokusil data změnit, by neměl dostat potvrzující zprávu na nově načtené stránce, ale spíše chybovou hlášku (tato chyba bude odstraněna opravením následujícího nálezu).

N13: Možnost nahlédnout na data uživatele po jeho odhlášení (závažnost střední)

Uživatel se přihlásí do aplikace, vyplní údaje ohledně zpracování osobních a marketingových údajů a údaje potvrdí stisknutím tlačítka Potvrdit. Tímto krokem by mělo dojít k odhlášení tohoto uživatele. „Jiný“ uživatel – útočník - stiskne v prohlížeči tlačítko zpět. Dojde k načtení stránky mainpage.html a útočník změní údaje a E-mail své rozhodnutí také potvrdí. Chybou je, že útočník po odhlášení uživatele může nahlédnout na rozhodnutí předchozího uživatele, nicméně náhledem do databáze bylo zjištěno, že změněné údaje se nepropsaly.

Testování správy relace

Při poslání požadavku na server, se v prohlížeči, rozhraní pro vývojáře (Developer Tools Interface) v sekci síť v podsekci cookies, dá vyčíst řada informací. Postupnými testy byl identifikován následující nález zahrnující ale více aspektů nastavení relace a cookies.

[87] V tomto momentě se tedy nejedná o čistý black box test.

N14: Nedostatečné nastavení správy relací (závažnost vysoká)

V prohlížeči bylo zjištěno, že je správa relací konstruována kódem v PHP, neboť je zde uvedená proměnná PHPSESSID. Testováním bylo dále zjištěno, že po zavření prohlížeče a opětovném otevření a načtení testované stránky dochází ke změně PHPSESSID. Pokud se uživatel přihlásí a na stránce mainpage.html nevykonává žádnou aktivitu po dobu cca. 5 minut a posléze vyplní souhlas a E-mail a stiskne tlačítko „Potvrdit“, dojde k načtení přihlašovací stránky a vypsaná data uživatelem nejsou v databázi uložena, což je v pořádku. Nicméně proměnná PHPSESSID se nemění ani potom, co se např. jiný uživatel v nevypnutém prohlížeči znovu přihlásí. Relace není obnovována, což může umožnit útoky na fixaci relace, kdy útočník donutí uživatele užít cookies známou útočníkovi a útočník by mohl dále ukrást relaci (OWASP, 2013, str. 92).

Testování validace vstupů a testy na straně klienta

Testy provedené v této oblasti se zaměřily na validaci vstupů v aplikaci a ověření, zda nejde zadané hodnoty při zasílání na server manipulovat. Dále bylo cílem ověřit možnost uskutečnit SQL injekci či XSS útok. Byly identifikovány následující nálezy:

N15: Absence validace na straně serveru (závažnost vysoká)

V některých polích jsou vstupy validovány na straně klienta – např., zda je zadán správný formát emailové adresy (jestli obsahuje znak @ a neobsahuje nepovolené speciální znaky, zda není adresa moc dlouhá) nebo zda jsou vyplněna všechna požadovaná pole. Povinnost vyplnit hodnoty je vynucována u kontaktního formuláře a také např. je dána povinnost vyplnit emailovou adresu na stránce mainpage.html, jestliže uživatel souhlasí se zpracováním údajů. V testech byl využit nástroj Burp Suite. Uživatel vyplní testovaná pole správně, ale při „cestě na server“ je požadavek pozastaven a v sekci Repeater (opakovač) upraven a poté na server přeposlán. Ukázalo se, že je možné uložit hodnoty, které podle frontendových validací nebyly přípustné – např. speciální znaky. S tím souvisí i následující nálezy.

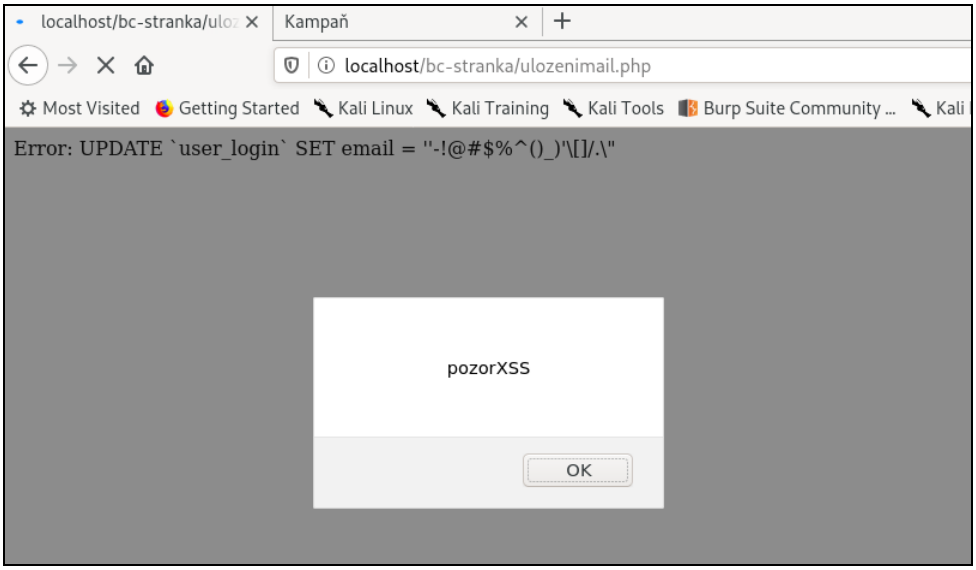
N16: Aplikace umožňuje XSS útok (závažnost kritická)

Při testování, jakým způsobem jsou validována pole, bylo zjištěno, že při zadání sady neplatných znaků se vypisuje na stránku chyba databáze (tato bude popsána v nálezu

dále), která v části popisu chyby cituje, co zadal uživatel. Této chyby bylo využito následujícím způsobem: v nástroji Burp Suite byla hodnota v poli E-mail a vyjádření souhlasu (radio button, který nabývá hodnot YES nebo NO) v požadavku upravena na hodnotu obsahující skript:

approval=yes&emailp=’ -!@#$%^()_)'\[]/.\"**<script>alert('pozorXSS')</script>**

Poté byl požadavek poslán na server. Stránka se načetla s JavaScriptovým alertem (viz obrázek č. 11), což značí, že aplikace je zranitelná na XSS útok. Útočník by mohl tímto způsobem spustit škodlivý kód.

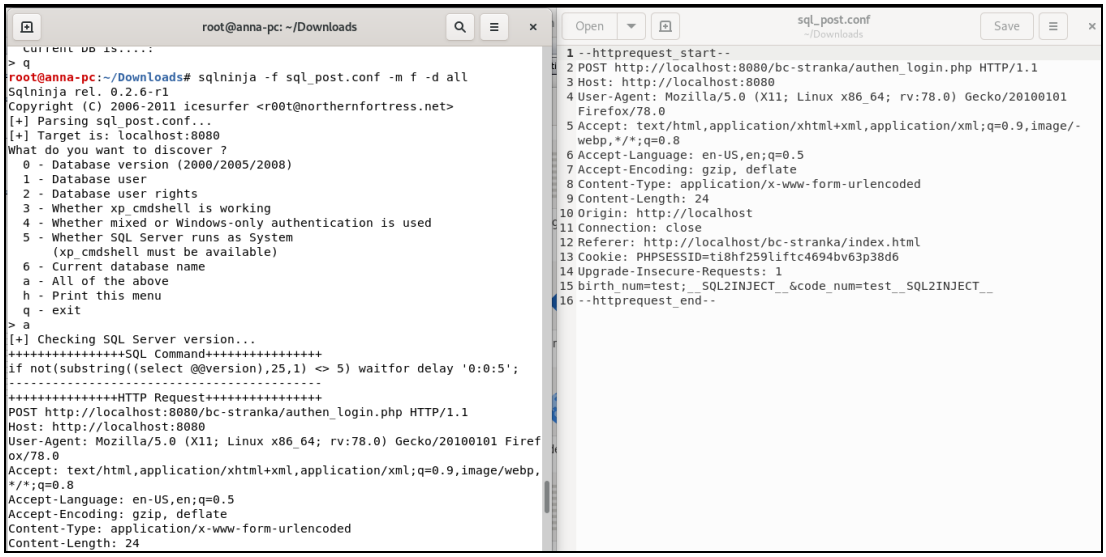


Obrázek 11: Demonstrace zranitelnosti na XSS útok

N17: Aplikace je náchylná na SQL injekce (závažnost kritická)

Existuje řada automatizovaných nástrojů, které umožňují zmapovat, jestli je aplikace zranitelná na SQL injekci. Autorka práce využila nástroje bez uživatelského rozhraní Sqlninja, který je automaticky k dispozici v distribuci Kali Linux. Prvním krokem, aby bylo možné spuštění, je definování konfiguračního souboru, kam se vloží požadavek zasílaný na server. Místa, která chceme injektovat je nutné označit __SQL2INJECT__. Po prvotním testu, který indikuje, zda je aplikace potenciálně zranitelná, je možné spustit tzv. Fingerprint útočný režim, který zjistí základní detaily o databázi a verzi, povahu uživatele přistupujícího do databáze, zda je xp_cmdshell pro uživatele k dispozici atd. Na obr. č. 12 je na pravé straně zobrazen konfigurační soubor pro

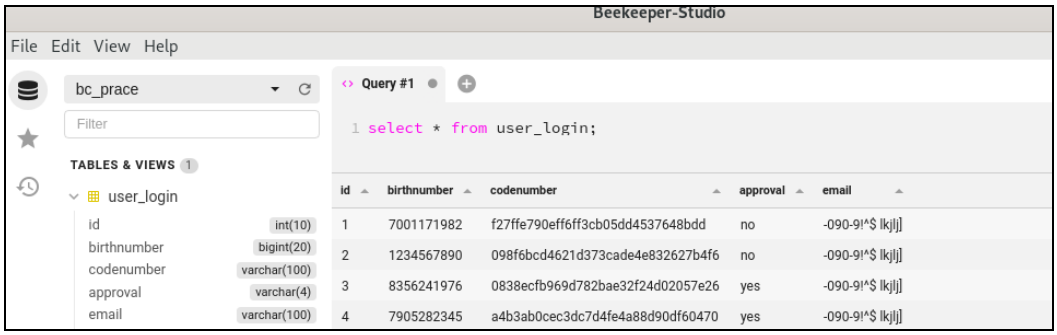
útok na přihlašovací pole aplikace a na levé straně spuštění Fingerprint skenu v příkazové řádce[88].



Obrázek 12: Spuštění testu v Sqlninja

Podle testu došlo ke zjištění, že je přistupováno k databázi v administrátorské roli, že xp_cmdshell je k dispozici atd. Toto však vychází z vědomého aktuálního nastavení a spuštění aplikace na lokální úrovni a nebude tudíž pokračováno v testech pomocí tohoto nástroje.

Pomocí manuálních testů přímo v aplikaci spočívajících v postupném zadávání hodnot do pole E-mail na stránce mainpage.html a úprav požadavků se např. podařilo zmanipulovat hodnoty E-mail a nastavit je pro všechny uživatele (viz obrázek č.13 níže).



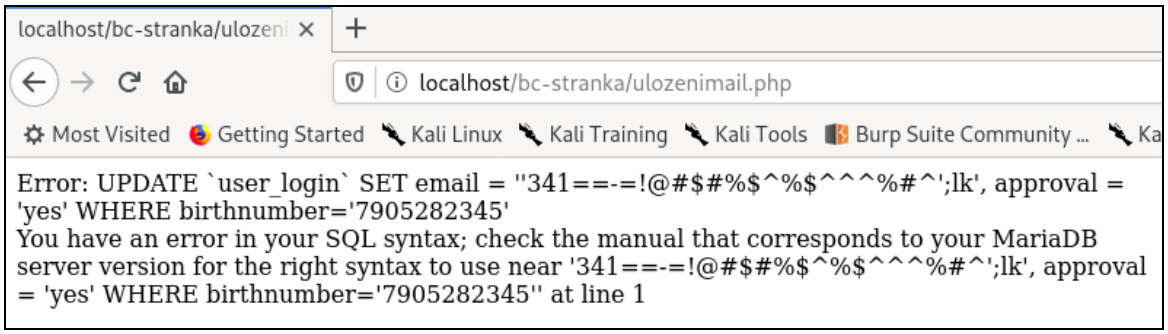
Obrázek 13: Změna hodnot v databázi u všech uživatelů (E-mail)

[88] Příkaz sqlninja -f sql_post.conf -m f -d all umožní spuštění režimu Fingerprint včetně vypsání požadavků poslaných na server a odpovědí serveru.

Testování chybových hlášek

N18: Neošetření chybových hlášek (závažnost vysoká)

Nálezy N15, N16 a N17 byly zjištěny a útoky provedeny i díky tomu, že bylo možné vyprovokovat chybovou hlášku prozrazující informace o databázi a způsobu, jak je formulován dotaz do databáze, jak se tabulka databáze jmenuje a jaké jsou její 3 sloupce. Tato chybová hláška byla vyvolána zadáváním různých kombinací speciálních znaků do pole pro E-mail na stránce mainpage.hmtl, se kterou je spojen PHP-kód ulozenimail.php.



Obrázek 14: Chybová hláška prozrazující informace o databázi

Testování obchodní (business) logiky

N19: Uložení E-mailu uživatele v případě nesouhlasu (závažnost nízká)

V aplikaci se přihlášený uživatel rozhoduje, udělí-li souhlas se zpracováním údajů. V případě, že uživatel zvolí nejprve volbu Souhlasím a vyplní E-mail, ale ihned poté názor změní a zatrhne variantu Nesouhlasím a poté stiskne tlačítko „Potvrdit“, pak i přes nesouhlas nalezneme v databázi vyplněný E-mail. K tomu by logicky nemělo dojít, neboť nepřeje-li si klient, aby došlo k zpracování údajů a kontaktování pro marketingové účely, neměl by být E-mail uložen.

5 Zhodnocení a doporučení

5.1 Zhodnocení provedených testů

Během testování v definovaném rozsahu došlo k identifikaci 19 bezpečnostních nálezů (seznam všech nálezů se nachází v příloze č. 1). Nálezy byly formulovány během testů, které patří do fáze implementace a do fáze verifikace. Přičemž fáze verifikace následovala bezprostředně po fázi testování pomocí statické analýzy kódu prostřednictvím nástroje SonarQube. Z toho vyplývá, že některým chybám, které byly nalezeny v konečné

fázi testování, se dalo jejich odstraněním ještě při vývoji předejít (např. odstranění chybové hlášky N3).

Testovaná aplikace není připravena k použití v produkčním prostředí vzhledem k povaze a závažnosti identifikovaných nálezů (zejm. těch o závažnosti kritické a vysoké), které je nutné bezprostředně ošetřit. Na následujících stránkách jsou popsána doporučení, jak zranitelnostem předejít a jak je odstranit.

5.1.1 Doporučení pro nápravu v oblasti nedostatků zdrojového kódu

N1: SQL dotazy založené na uživatelem kontrolovaných datech (závažnost kritická)

Řešením identifikovaného problému by bylo spíše využít připravených příkazů (než použití spojených řetězců), aby došlo k „escapování“[89] uživatelem zadaných dat a škodlivé hodnoty by neovlivnily nevhodným způsobem dotaz do databáze. Dalším řešením by mohlo být validovat každý parametr dotazu nebo validovat uživatelem vložené hodnoty oproti seznamu povolených hodnot. Také jsou u výsledků analýzy doporučeny databázové ORM (SonarCloud, 2020). Objektově relační mapování umožňuje mapování relační databáze na objekty programovacího jazyka. Vývojáři mohou využít v této oblasti i existující frameworky pro jazyk PHP, kterými jsou např. Doctrine nebo Propel. U malých aplikací však může využití komplexního frameworku ztrácet význam (Tröster, 2010).

N2: Chybějící ochrana databáze prostřednictvím hesla (závažnost kritická)

Místo v kódu bylo ze strany SonarQube skutečně správně identifikované jako místo, kde bývá uvedeno heslo pro databázi. Je zde prázdný řetězec, který indikuje, že heslo nebylo nastaveno a tudíž není databáze chráněna. Heslo však bylo autorkou kódu před zveřejněním kódu v repozitáři GitHub manuálně odstraněno a databáze je heslem ve skutečnosti chráněna. Jistě si však zaslouží i toto místo lepší ošetření dané situace: např. užitím proměnných, které jsou nakonec uložené v složce gitignore. Tedy definování místa, kde uložené soubory nejsou sledované a přístupné.

N3: Nevhodné reflektování uživatelem kontrolovaných dat (závažnost kritická)

Řešením je i v tomto případě použít validaci vstupu zadaného uživatelem oproti seznamu povolených hodnot či zbavit vstupní data znaků sloužících ke škodlivým účelům

[89] Použití únikového znaku.

(zde je možné využít knihoven vytvořených speciálně pro bezpečnostní účely). Možné je také zakódovat vstupy uživatele a upravit zakódovaný výstup vzhledem ke kontextu, ve kterém je používaný (SonarSource, 2020).

N4: Slabý hašovací algoritmus (závažnost vysoká)

Doporučuje se použít namísto MD5 jiné algoritmy jako např. SHA-256, SHA-512, SHA-3 nebo bcrypt. Pro hašování hesel je dokonce doporučeno použít algoritmus bcrypt (místo SHA-256), který se nepočítá příliš rychle, protože pak zpomaluje brute force i slovníkový útok (SonarCloud, 2020).

5.1.2 Doporučení k odstranění nálezů ve fázi verifikace

N5: Nenastavení X-Frame volby v hlavičce odpovědi (závažnost střední)

Nejmodernější prohlížeče podporují X-Frame možnosti v hlavičce HTTP. Je potřeba toto správně nastavit na všech stránkách aplikace. Doporučovanou variantou je nastavit v konfiguraci webserveru tuto možnost na hodnotu SAMEORIGIN (stránka může být načtena z iframu, pokud obě stránky mají stejný původ) nebo DENY, což způsobí zamítnutí a stránka nemůže být načtena z iframu. (Kofler a kol., 2019, str. 781).

N6: Absence tokenu proti CSRF útoku (závažnost nízká)

Lze doporučit použít prověřené knihovny nebo frameworky, které tuto zranitelnost neumožní, a to tedy již ve fázi designu (existuje např. anti-CSFR balíček od společnosti OWASP). ZAP také doporučuje použít nástroje po správu relací, jejichž součástí jsou i komponenty proti CSRF (Zaproxy, 2020).

Pokud jde o konkrétní opatření v kódu, pak přidání tokenu v praxi bude znamenat, že webserver nastaví tento token do cookies hned potom, co se uživatel přihlásí. Každé odeslání dat formulářem obsahuje skryté pole s tokenem, čímž se zranitelnost CSRF odstraní, neboť podvrhnout autenticitu odeslaných dat, např. za uživatele není bez znalosti tokenu možné. Server ověřuje, zda požadavek na něj zaslaný token obsahuje a pokud nikoliv, nebude proveden. Musí se samozřejmě generovat token, který se nedá předvídat a jehož platnost vyprší po krátkém čase. Kód musí také verifikovat, zda obdržený token je stejný, jako ten nastavený. CSRF token by se také neměl posílat GET požadavkem, aby nebyl dostupný přímo v URL. Příklad pro generování tokenu v PHP je: $_SESSION['token'] = bin2hex(random_bytes(24)). Pro vyšší zabezpečení lze používat

různé tokeny pro každý požadavek, což spočívá v zneplatnění použitého tokenu ihned potom, co byl verifikován (Netsparker, 2020).

N7: Cookie bez HttpOnly flagu (závažnost nízká)

Řešením této chyby je nastavit tento flag HttpOnly pro všechny cookies do Set-Cookie HTTP odpovědní hlavičky serveru. Např.:
“Set-Cookie: <name>=<value>[; <Max-Age>=<age>]
`[; expires=<date>][; domain=<domain_name>]
[; path=<some_path>][; secure][; HttpOnly].”
(OWASP, 2020).

N8: Cookie bez atributu SameSite (závažnost nízká)

Je zapotřebí správně nastavit SameSite atribut u Set-Cookie HTTP odpovědní hlavičky serveru. Atribut přijímá tři hodnoty: Lax, Strict a dále None, která umožňuje posílání cookies ve všech kontextech, což není žádoucí. Vhodné je nastavení omezit na Strict, při kterém nebudou soubory cookie odesílány v kontextu požadavků zasílaných třetími stranami. V nastavení Lex je možné posílat cookies v rámci navigace na nejvyšší úrovni a pomocí GET požadavků (MDN, 2020, Zaproxy, 2020). Při nastavování tohoto atributu u testované aplikace je nejlepší variantou použít hodnotu Strict, protože neobsahuje žádné iframy ani linky na jiné stránky, kde by bylo nutné použít např. Lax.

N9: Chybějící nastavení X-Content-Type-Options (závažnost nízká)

Obranou proti MIME-sniffingu a na něj navazujícími jinými útoky je správné nastavení X-Content-Type-Options na hodnotu „nosniff“ pro všechny stránky (Zaproxy, 2020).

N10: Absence zamykacích mechanismů při přihlášení (závažnost vysoká)

Doporučením je nastavit mechanismus zamknutí účtu po zadání neplatných přihlašovacích údajů. Zpravidla má jít při nastavení mechanismů zamykání o vyvážený systém, který je dostatečně bezpečný, ale ještě uživatelsky přívětivý. Vhodné by však bylo nějaký limit pro špatná přihlášení nastavit: zpravidla se doporučuje po třech až pěti špatných pokusech zamknout účet uživateli a odemknout ho po určité stanovené době (OWASP, 2013, str. 72).

N11: Aplikace není chráněna před automatizovanými útoky (závažnost kritická)

Doporučovaným řešením, jak předcházet automatizovaným útokům, je např. použít systém CAPTCHA[90]. Cílem CAPTCHA je testovat, zda je uživatel člověk nebo automatizovaný nástroj. Implementace takového typu ověření můžou být však taktéž zranitelné. Doporučuje se užívat ověřené implementace jako např. Google ReCaptcha.

Dalším řešením je implementovat vícestupňové ověření (Multi-Factor Authentication), ve kterém jde o vyžádání dalšího vstupu nebo akce uživatele při autentizaci. Automatizovanému útoku jde také zabránit zamezením posílat požadavky z určité IP adresy po několika opakovaných útocích za jednotku času. Je také možné zpřístupnit zasílání požadavků jen z určitých povolených IP adres (whitelisting), dále zpomalit odpovědi serveru po větším počtu požadavků při neúspěšném přihlášení (Melamed, 2017).

N12: Absence protokolu HTTPS (závažnost kritická)

Je zapotřebí implementovat HTTPS namísto existujícího HTTP za účelem zabezpečení komunikace. Vhodné je také nastavit na webserveru HSTS, např. v konfiguračním souboru v httpd.conf:

Header always set Strict-Transport-Security „max-age=31536000“; includeSubDomains.

Také je v tomto konfiguračním souboru nutné nastavit (trvalé) přesměrování z HTTP na HTTPS stránku (Kofler, 2019, str. 784).

N13: Možnost nahlédnout na data uživatele po jeho odhlášení (závažnost střední)

Po tom, co dojde k odhlášení uživatele stisknutím tlačítka „Zrušit“ (a načte se poté úvodní stránka) nebo „Potvrdit“ (a načte se poté potvrzovací stránka) a následném stisknutí tlačítka zpět, by nemělo dojít k načtení dat uživatele. Tomuto lze zabránit několika způsoby. Jedním z nich je v kódu upravit vynucené vyčištění cache nebo použít meta tag no-cache. Je také možné nechat vyčistit historii prohlížeče a přesměrovat stránku, což ale nemusí být uživatelsky přívětivým řešením, neboť ne každý uživatel stojí o promazání historie po odhlášení z aplikace (MEMBER10510822, 2013).

[90] CAPTCHA je zkratkou pro Completely Automated Public Turing test to tell Computers and Humans Apart.

N14: Nedostatečné nastavení správy relací (závažnost vysoká)

Doporučuje se obnovit (regenerovat) ID relace během procesu autentizace, protože se zde mění status anonymního uživatele v uživatele ověřeného. Další situací vhodné ošetření je odhlášení, kde se mění status uživatele ověřeného v anonymního. V PHP se jedná o následující kód: session_regenerate_id(true). Další možností, jak zpřehlednit tuto situaci a umožnit uživateli rozhodnout jednoznačně o odhlášení by bylo zabudovat do aplikace tlačítko „Odhlásit“, ve spojení s kterým by došlo k regenerování ID relace a zničení údajů – proměnných souvisejících s relací. Vzhledem k tomu, že aplikace je navržena tak, že obsahuje pouze jednu stránku, po které je uživatel informován o odhlášení po potvrzení zvoleného rozhodnutí, bylo to autorkou aplikace vyhodnoceno, že tato úprava není nutná.

V kódu je dále možné nastavit, za jaký čas vyprší relace, např. nastavením Max-Age atributu u cookies. Čím kratší je interval relace, tím menší je pravděpodobnost zneužití relace útočníkem. U kritičtějších aplikací se má používat vypršení po 2-5 minutách a u méně kritických může jít o 15-30 minut (CheatsheetSeries, 2020).

Na zvážení je také změnit název proměnné PHPSESSID tak, aby nedošlo k prozrazení užité technologie.

N15, N16: Absence validace na straně serveru; možný XSS útok (závažnost vysoká)

Řada webových aplikací, nejen ta, která je nyní objektem testování, obsahuje pouze validace na straně klienta. Jak bylo demonstrováno při testování i v našem případě se dá frontendová validace snadno obejít použitím proxy nástroje mezi prohlížečem a webserverem. Řešením je zajistit validaci hodnot vepsaných do polí nejen na straně klienta, ale hlavně na straně serveru (Kofler a kol., 2019, str. 793).

N17: Aplikace je náchylná na SQL injekce (závažnost kritická)

Kromě výše uvedeného (doporučení pro N15 a N16) platí pro SQL injekci další specifická doporučení. Jedním z nich je parametrizování SQL dotazů, při kterých se definuje SQL dotaz a použijí se zástupné symboly (placeholders) pro uživatelem poskytnuté proměnné, což umožní rozeznat mezi SQL dotazem a daty poslanými uživatelem. Metodou, jak tento postup ještě zjednodušit, je použít PDO (PHP Data Object), které parametrizování dotazů usnadní (Muscat, 2019).

N18: Neošetření chybových hlášek (závažnost vysoká)

Je potřeba konstrukci chybové hlášky upravit tak, aby nedošlo k vyzrazení uvedených informací o aplikaci (databázi). Hlášení o chybě musí být obecné.

N19: Uložení E-mailu uživatele v případě nesouhlasu (závažnost nízká)

Řešením je upravit aplikační logiku tak, aby v případě nesouhlasu uživatele se zpracováním údajů nedošlo k uložení E-mailu v databázi.

6 Závěr

I přes velmi komplexní povahu tématu bezpečnostního testování webových aplikací se podařilo popsat v teoretické části základní termíny, místo bezpečnostního testování ve vývojovém cyklu softwaru a dále upozornit na nejaktuálnější rizika, která webových aplikacím hrozí. Byly představeny metodologie a způsoby, jak webovou aplikaci otestovat z bezpečnostního hlediska. Ukázalo se že metodologie OWASP Testing Guide pokrývá velmi prakticky celou škálu testů od zjišťování informací, přes testování ověření, validace na straně klienta a serveru, po obchodní logiku aplikace.

V praktické části došlo na základě nasbíraných poznatků k testování autorkou vytvořené webové aplikace nazvané Kampaň. Vzhledem k tomu, že provedení všech bezpečnostních testovacích aktivit pokrývajících celý cyklus vývoje softwaru by značně přesahoval rámec práce, byly vybrány 2 typy testů. Jednalo se o bezpečnostní testování v rámci implementace softwaru (automatická statická analýza kódu zaměřená na prvky bezpečnosti) a testování ve fázi vývojového cyklu zvané verifikace, za využití výše popsané metodologie OWASP. V této fázi šlo zejména o tzv. black box testy. Limitem těchto testů je skutečnost, že byly provedeny 1 osobou: autorkou práce co by vývojářskou aplikace a zároveň testerkou, což nepředstavuje klasickou testovací situaci.

Při testech se podařilo identifikovat bezpečnostní nálezy a následně doporučení k odstranění identifikovaných chyb. Na základě nálezů bylo také možné učinit závěr o testované aplikaci, a to, zda může být v daném stavu nasazena na produkci. Závěr byl negativní – nejprve je nutné opravit nalezené chyby, než bude aplikace používána uživateli v běžném provozu.

Přesto, že se jednalo o poměrně jednoduše konstruovanou aplikaci, je nutné konstatovat, že 19 nálezů (z nichž více bylo kritické povahy), ukazují na skutečnost, že i v tak malém rozsahu kódu se může vývojář dopustit mnoha chyb, které by mohly mít v případě nasazení produktu do produkce velmi negativní následky, zejm. pokud by šlo o kritičtější aplikaci.

Na internetu je možné získat řadu návodů, jakým způsobem tu či onu komponentu aplikace v programovacím jazyce napsat. Většina z nich však postrádá popsání elementárních bezpečnostních principů při vývoji. Po zkušenosti z vývoje testované aplikace a také po vyhodnocoení nálezů se nabízí doporučení, zejm. pro ne-seniorní

vývojáře, využít při vývoji, alespoň pro komponenty týkající se autentizace, některý z ověřených existujících frameworků, který nastaví i základní principy pro správu relací.

Pozitivně lze hodnotit, že dnes je vývojářům i testerům k dispozici škála nástrojů v komunitních verzích (zdarma). Příkladem byl SonarQube pro vývojáře pro statickou analýzu kódu a OWASP ZAP, Burp Suite a další nástroje pro testery a bezpečnostní experty, které posloužily pro testy ve fázi verifikace. Některé nástroje (např. SonarQube, OWASP ZAP) obsahují kromě identifikace chyb i odůvodnění, proč je daný nález rizikový, a doporučení, jak chybu opravit (někdy více, jindy méně konkrétní).

Ukázalo se dále, že nelze spoléhat pouze na automatizované testy. Tyto sice objeví mnoho potenciálních chyb, často spadajících do kategorie konfigurace, ale je nutné je doplnit i testy manuálními. Manuální testy umožnily objevit chyby, které automatizovaný nástroj neobjevil, vzhledem ke komplikovanější povaze testu nebo nemožnosti automatizovaných testů objevit chybu v logice aplikace. Manuální testy dále posloužily k verifikaci zranitelností objevených automatizovaným testem.

Testy provedené ve fázi vývoje bezprostředně předcházely testům ve fázi verifikace. Tzn., že pokud by se nálezy identifikované v rámci statické analýzy kódu ihned opravily, předešlo by se množství nálezů v pozdější fázi verifikace. Toto potvrzuje nutnost začít s aktivitami a nastavením základních mechanismů v oblasti bezpečnosti v nejrannějších fázích vývoje softwaru a průběžně nalezené chyby opravovat. Při průchodem vývojovým cyklem softwaru by toto opatření mělo vést k vytvoření bezpečnějšího produktu a k úsporám ohledně oprav chyb identifikovaných pozdě ve vývoji a v horším případě v době, kdy už je produkt produkčně využíván; o dalších negativních důsledcích, jako je např. ztráta dat, poškození reputace atd., ani nemluvě.

7 Seznam použitých zdrojů

1. CVE. *About CVE* [online]. Mitre, 2020 [cit. 2020-08-18]. Dostupné z: <https://cve.mitre.org/about/index.html>.
2. CWE. *Overview - What Is CWE?* [online]. Mitre, 2020 [cit. 2020-08-18]. Dostupné z: <https://cwe.mitre.org/about/index.html>.
3. ČSN ISO 31000 (010351). Management rizik - Principy a směrnice. 2018.
4. ČSN ISO 27001 (369797). Management bezpečnosti informací. 2014.
5. BECK, Kent, BEEDLE, Mike, BENNEKUM, Arie van, CUNNINGHAM, Ward, FOWLER, Martin, GRENNING, James, HIGHSMITH, Jim, HUNT, Andrew, JEFFRIES, Ron, KERN, Jon, MARICK, Brian, MARTIN, Robert C., MELLOR, Steve, SCHWABER, Ken, SUTHERLAND, Jeff, THOMAS, Dave. *Manifest Agilního vývoje software* [online]. 2001 [cit. 2020-08-03]. Dostupné z: < http://agilemanifesto.org/iso/cs/manifesto.html>.
6. DINITA, Madalina. *US Homeland Security warns you to stop using IE entirely* [online]. Windows Report, 2020 [cit. 2020-03-03]. Dostupné z: <https://windowsreport.com/internet-explorer-security-issues/>.
7. EXPLOIT-DB. *About The Exploit Database.* [online]. Offensive Security, 2020 [cit. 2020-07-05]. Dostupné z: <https://www.exploit-db.com/>.
8. FIRST. *Common Vulnerability Scoring System SIG.* [online]. FIRST, 2020 [cit. 2020-08-18]. Dostupné z: <https://www.first.org/cvss/>.
9. HACKERONE. *The 4th Hacker Powered Security Report* [online]. 100 s. (PDF). Hacker One, 2020 [cit. 2020-10-18]. Dostupné z: <https://www.hackerone.com/resources/latest-news-insights/the-4th-hacker-powered-security-report>.
10. HERZOG, Pete. *The Open Source Security Testing Methodology Manual* [online]. Verze 3. 213 s. (PDF). The Institute for Security and Open Methodologies (ISECOM), 2010 [cit. 2020-08-03]. Dostupné z: <https://www.isecom.org/OSSTMM.3.pdf>.
11. HOWARD, Michael, PINCUS, Jon, WING, Jeannette M. *Measuring Relative Attack Surfaces* [online]. 15 s. (PDF). 2003, [cit. 2020-07-28]. Dostupné z: <http://www.cs.cmu.edu/afs/cs/project/svc/projects/security/wadis1.pdf>.

12. HSU, Tony, 2018. *Hands-On Security in DevOps - Ensure continuous security, deployment, and delivery with DevSecOps.* Brimingham: Packt Publishing Ltd. 341 s. ISBN 978-1-78899-550-4.
13. CHEATSHEETSERIES. *Session Management Cheat Sheet* [online]. OWASP, 2020, [cit. 2020-11-01]. Dostupné z: <https://cheatsheetseries.owasp.org/cheatsheets/Session_Management_Cheat_Sheet.html>.
14. ICESURFER, NICO. *Sqlninja user manual* [online]. Sqlninja, 2020, [cit. 2020-10-28]. Dostupné z: <http://sqlninja.sourceforge.net/sqlninja-howto.html>.
15. ISTQB. *Certified Tester Foundation Level Syllabus* [online]. Verze 2018 V3.1. 93 s. (PDF). International Software Testing Qualifications Board, 2019 [cit. 2020-03-12]. Dostupné z: <https://www.istqb.org/downloads/send/2-foundation-level-documents/281-istqb-ctfl-syllabus-2018-v3-1.html>.
16. KALI. *What is Kali Linux?* [online]. Offensive Security, 2020 [cit. 2019-10-24]. Dostupné z: <https://www.kali.org/docs/introduction/what-is-kali-linux/>.
17. KANER, Cem, BACH, James, PETTICHIRD, Bret, 2001. *Lessons learned in Software Testing*. New York: WILEY. 320 s. ISBN 0-471-08112-4.
18. KHAWAJA, Gus, 2018. *Practical Web Penetration Testing: Secure web applications using Burp Suite, Nmap, Metasploit, and more.* Brimingham: Packt Publishing Ltd. 283 s. ISBN 978-1-78862-403-9.
19. KISSEL, Richard, STINE, Kevin, SCHOLL, Matthew. *Security Considerations in the System Development Life Cycle Assessments* [online]. Rev 2. 68 s. (PDF). NIST Special Publication 800-64, 2018 [cit. 2019-10-25]. Dostupné z: <https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-64r2.pdf>.
20. KOFLER, Michael, ZINGSHEIM, André, GEBESHUBER, Klaus, WIDL, Markus, AIGNER, Roland, HACKNER, Thomas, KANIA, Stefan, KLOEP, Peter, NEUGEBAUER, Frank, 2019. *Hacking & Security, Das umfassende Buch*. Bonn: Rheinwerk Verlag GmbH. 1067 s. ISBN 978-3-8362-4548-7.
21. KOTHE, Priyanka. *Fuzz Testing (Fuzzing) Tutorial: What is, Types, Tools & Example* [online]. Guru99, 2020 [cit. 2020-05-12]. Dostupné z: <https://www.guru99.com/fuzz-testing.html?fbclid=IwAR0397B1fOWAmmRm0Tri6Jnra8buGyOWo00cKqaPQ2w93OIiyo2AxB2sshs>.

22. KUROSE, James F., ROSS, Keith W., 2014. *Počítačové sítě*. Brno: Computer Press. 622 s. ISBN 978-80-251-3825-0.
23. MARSHALL, Joseph, 2018. *Hands-On Bug Hunting for Penetration Testers*. Brimingham: Packt. 234 s. ISBN 978-1-78934-420-2.
24. MDN Web Docs. *Resources for developers, by developers* [online]. Mozilla, 2013 [cit. 2020-10-23]. Dostupné z: <https://developer.mozilla.org/en-US>.
25. MELAMED, Tal. *Brute Force Prevention* [online]. AppSec, 2017 [cit. 2020-10-21]. Dostupné z: <https://appsec-labs.com/portal/brute-force-attack-part-2/>
26. MEMBER10510822. *Browser Back Button Issue After Logout* [online]. CodeProject.com, 2013 [cit. 2020-10-23]. Dostupné z: <https://www.codeproject.com/Tips/549347/Browser-Back-Button-Issue-After-Logout-2>.
27. MEUCCI, Matteo, MULLER, Andrew. *OWASP Testing Guide* [online]. Verze 4. 224 s. (PDF). 2013 [cit. 2020-05-23]. Dostupné z: <https://owasp.org/www-project-web-security-testing-guide/assets/archive/OWASP_Testing_Guide_v4.pdf>.
28. MICROSOFT. *Microsoft Security Development Lifecycle (SDL) Process Guidance* [online]. Verze 5.2. 168 s. (DOC). 2012 [cit. 2020-05-23]. Dostupné z: <https://www.microsoft.com/en-us/download/details.aspx?id=29884>.
29. MILLER, Bart. *Project List* [online]. 3. s. (PDF). Computer Sciences Department University of Winsconsin-Madison, 1988 [cit. 2020-05-23]. Dostupné z: <http://pages.cs.wisc.edu/~bart/fuzz/CS736-Projects-f1988.pdf>.
30. MUSCAT, Ian. *How to Prevent SQL Injection Vulnerabilities in PHP Applications* [online]. Acunetix, 2019 [cit. 2020-10-20]. Dostupné z: <https://www.acunetix.com/blog/articles/prevent-sql-injection-vulnerabilities-in-php-applications>.
31. NETCRAFT. *October 2020 Web Server Survey List* [online]. Netcraft, 2020 [cit. 2020-10-27]. Dostupné z: <https://news.netcraft.com/archives/category/web-server-survey/>.
32. NETSPARKER Security Team. *How to Protect Your Website Using Anti-CSRF Tokens* [online]. Netsparker, 2020 [cit. 2020-10-20]. Dostupné z: <https://www.netsparker.com/blog/web-security/protecting-website-using-anti-csrf-token>.

33. NIST. *National Vulnerability Database* [online]. NIST, 2020 [cit. 2020-10-20]. Dostupné z: <https://nvd.nist.gov/>.
34. NTTSECURITY. *Global Threat Intelligence Report* [online]. Verze 2. 50 s. (PDF). NTT Security, 2019 [cit. 2020-10-20]. Dostupné z: <https://www.nttsecurity.com/docs/librariesprovider3/resources/2019-gtir/2019_gtir_report_2019_uea_v2.pdf>.
35. OWASP. *Application Security Verification Standard* [online]. Verze 4.0.2. 69 s. (PDF). OWASP, 2020 [cit. 2019-11-01]. Dostupné z: <https://raw.githubusercontent.com/OWASP/ASVS/v4.0.2/4.0/OWASP%20Application%20Security%20Verification%20Standard%204.0.2-en.pdf>.
36. OWASP. *OWASP Top Ten – 2017, The Ten Most Critical Web Application Security Risks* [online]. 25 s. (PDF). OWASP, 2017 [cit. 2019-08-18]. Dostupné z: <https://owasp.org/www-pdf-archive/OWASP_Top_10-2017_%28en%29.pdf.pdf>.
37. OWASP. *OWASP SAMM* [online]. 2020 [cit. 2019-08-18]. Dostupné z: < https://owasp.org/www-project-samm/>.
38. OWASP. *OWASP Code Review Guide* [online]. Verze 2.0. 220 s. (PDF). OWASP, 2017 [cit. 2020-07-10]. Dostupné z: <https://owasp.org/www-pdf-archive/OWASP_Code_Review_Guide_v2.pdf>.
39. PCI, PTGSIP. *Penetration Testing Guidance* [online]. Verze 1.1. 44 s. (PDF). PCI Security Standards Council, 2017 [cit. 2020-09-18]. Dostupné z: <https://www.pcisecuritystandards.org/documents/Penetration-Testing-Guidance-v1_1.pdf?agreement=true&time=1603999751440>.
40. PORTSWIGGER. *OS command injection* [online]. PortSwigger, 2020 [cit. 2020-08-19]. Dostupné z: <https://portswigger.net/web-security/os-command-injection>.
41. RICE, Randall, DAUGHTREY, Hugh T., DIJKMAN, Frans, OLIVEIRA, Joel, RIBAULT, Alain. *Certified Tester Advanced Level Syllabus Security Tester* [online]. Verze 2016. 86 s. (PDF). International Software Testing Qualifications Board, 2016 [cit. 2020-03-12]. Dostupné z: <https://www.istqb.org/downloads/send/46-advanced-level-security-tester/194-advanced-security-tester-syllabus-ga-2016.html>.

42. RISKSENSE. *Cracks in the Foundation: Web and Application Framework Vulnerabilities* [online]. 22 s. (PDF). RiskSense Spotligth Report, 2020 [cit. 2020-05-01].
Dostupné z: <https://risksense.com/wp-content/uploads/2020/07/RiskSense-Spotlight-Report-Web-and-App-Frameworks.pdf>.
43. ROSS, Ronald S. *Guide for Conducting Risk Assessments* [online]. Rev 1. 95 s. (PDF). NIST Special Publication 800-30, 2012 [cit. 2020-05-25]. Dostupné z: <https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-30r1.pdf>.
44. ROSS, Ron, PILLITTERI, Victoria, GRAUBHART, Richard, BODEAU, Deborah, MCQUAID, Rosalie. *Developing Cyber Resilient Systems: A Systems Security Engineering Approach* [online]. 229 s. (PDF). NIST Special Publication 800-160 Volume 2, 2019 [cit. 2020-09-02]. Dostupné z:
<https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-160v2.pdf>.
45. SCARFONE, Karen, SOUPPAYA, Murugiah, CODY, Amanda, OREBAUGH, Angela. *Technical Guide to Information Security Testing and Assessment - Recommendations of the National Institute of Standards and Technology* [online]. 80 s. (PDF). NIST Special Publication 800-115, 2008 [cit. 2020-08-25]. Dostupné z:
<https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-115.pdf>
46. SHOSTACK, Adam, 2014. *Threat Modeling : Designing for Security*. John Wiley & Sons, Incorporated. 591 s. ISBN 978-1-118-80999-0.
47. SIMON, Frank, GROSSMANN, Jürgen, GRAF, Christian A., MOTTOK, Jürgen, SCHNEIDER, Martin A., 2019. *Basiswissen Sicherheitstests. Aus- und Weiterbildung zum ISTQB® Advanced Level Specialist – Certified Security Tester.* Heidelberg: dpunkt.verlag. 394 s. ISBN 978-3-86490-618-3.
48. SKF. *Training developers in writing secure code* [online]. OWASP, 2020 [cit. 2020-10-29]. Dostupné z:
<https://www.securityknowledgeframework.org/index.php>
49. SLADE, Robert, 2006. *Dictionary of Information Security*. Rockland MA: Syngress Publishing, Inc. 222 s. ISBN: 1597491152.
50. SONARCLOUD. *Why is this an issue?* [online]. SonarQube, 2020 [cit. 2020-10-20]. Dostupné z: <https://sonarcloud.io/project/issues?id=sarkamoes_lamp-webapp&resolved=false&types=VULNERABILITY>.

51. SONARSOURCE. *Endpoints should not be vulnerable to reflected cross-site scripting (XSS) attacks.* [online]. SonarQube, 2020 [cit. 2020-10-20]. Dostupné z: <https://rules.sonarsource.com/php/RSPEC-5131>.
52. SONARQUBE. *SonarQube Documentation.* [online]. SonarQube, 2020 [cit. 2020-09-02]. Dostupné z: <https://docs.sonarqube.org/latest/>.
53. TRÖSTER, František. *ORM frameworky pro PHP5: Obecný úvod* [online]. Zdrojak.cz, 2020 [cit. 2020-10-02]. Dostupné z: <https://www.zdrojak.cz/clanky/orm-frameworky-pro-php5-obecny-uvod>.
54. TSE, Ronald. *Information Security Management through Reflexive Security - Six Pillars in the Integration of Security, Development and Operations* [online]. 18 s. (PDF). Cloud Security Alliance, 2019 [cit. 2020-09-02]. Dostupné z: <https://cloudsecurityalliance.org/artifacts/information-security-management-through-reflexive-security>.
55. VERONA, Joakim, 2018. *Practical DevOps, Second Edition: Implement DevOps in your organization by effectively building, deploying, testing, and monitoring code.* Brimingham: Packt Publishing Ltd. 242 s. ISBN 978-1-78839-257-0.
56. W3TECHS. *Usage statistics of server-side programming languages for websites* [online]. W3Techs – Web Technology Surveys, 2020 [cit. 2020-10-27]. Dostupné z: https://w3techs.com/technologies/overview/programming_language>.
57. ZAPROXY. *OWASP Zed Attack Proxy (ZAP)* [online]. OWASP, 2020 [cit. 2020-10-24]. Dostupné z: https://www.zaproxy.org>.

8 Přílohy

Příloha č. 1: Seznam bezpečnostních nálezů v testované aplikaci 66
Příloha č. 2: CD se zdrojovým kódem.....66

8.1 Příloha č. 1: Seznam bezpečnostních nálezů v testované aplikaci

ID nálezu	Název nálezu	Závažnost
N1	SQL dotazy založené na uživatelem kontrolovaných datech	kritická
N2	Chybějící ochrana databáze prostřednictvím hesla	kritická
N3	Nevhodné reflektování uživatelem kontrolovaných dat	kritická
N4	Slabý hašovací algoritmus	vysoká
N5	Nenastavení X-Frame volby v hlavičce odpovědi	střední
N6	Absence tokenu proti CSRF útoku	nízká
N7	Cookie bez HttpOnly flagu	nízká
N8	Cookie bez atributu SameSite	nízká
N9	Chybějící nastavení X-Content-Type-Options	nízká
N10	Absence zamykacích mechanismů při přihlášení	vysoká
N11	Aplikace není chráněna před automatizovanými útoky	kritická
N12	Absence protokolu HTTPS	kritická
N13	Možnost nahlédnout na data uživatele po jeho odhlášení	střední
N14	Nedostatečné nastavení správy relací	vysoká
N15	Absence validace na straně serveru	vysoká
N16	Aplikace umožňuje XSS útok	kritická
N17	Aplikace je náchylná na SQL injekce	kritická
N18	Neošetření chybových hlášek	vysoká
N19	Uložení E-mailu uživatele v případě nesouhlasu	nízká

8.2 Příloha č. 2: CD se zdrojovým kódem