Vysoká škola ekonomická v Praze

Fakulta informatiky a statistiky

Katedra systémové analýzy

Bakalářská práce

2016 **Ondřej Doležal**

Vysoká škola ekonomická v Praze

Fakulta informatiky a statistiky

Katedra systémové analýzy

Testování bezpečnosti webové aplikace pro evidenci deficitů na silniční síti TEN-T a implementace oprav

Vypracoval: Ondřej Doležal

Vedoucí práce: Ing. Jaromír Veber, Ph.D.

Rok vypracování: 2016

Čestné prohlášení:

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Veškeré použité podklady, ze kterých jsem čerpal informace, jsou uvedeny v seznamu použité literatury a citovány v textu podle normy ČSN ISO 690.

V..... dne .... Podpis: .....

Abstrakt práce

Cílem práce je čtenářům představit bezpečnost webových aplikací jako součást vývoje, seznámit je s možnými hrozbami, které existují. Součástí práce je také zmínka o prostředcích, které lze pro testování využít. V neposlední řadě práce představuje aplikaci autora, na které je provedeno testování a následná praktická implementace oprav.

Abstract

The main objective of this bachelor thesis is to introduce the topic of web security as the part of development. This thesis mentions the most critical web application security risks and tools which developers may use to test such risks. Another objective of the thesis is to introduce a web application which this author co-developed and which is the subject of security testing and following implementation of fixes.

Obsah

1. Úvod 1

Cíle práce 2

2. Úvod do testování bezpečnosti 3

3. OWASP 5

O organizaci 5

OWASP TOP 10 8

A1 - Injektování 9

Obrana 11

A2 – Chybná autentizace a správa relace 12

Obrana 13

A3 – Cross-Site Scripting (XSS) 13

Obrana 14

A4–Nezabezpečený přímý odkaz na objekt 14

Obrana 15

A5– Nezabezpečená konfigurace 16

Obrana 16

A6 – Expozice citlivých dat 16

Obrana 17

A7 – Chyby v řízení úrovní přístupu 17

Obrana 18

A8 - Cross-Site Request Forgery (CSRF) 18

Obrana 19

A9 – Použití známých zranitelných komponent 20

Obrana 21

A10 – Neošetřené přesměrování a předávání 21

Obrana 21

OWASP Testing Guide 22

4. Aplikace CEBASS 23

Autoři **23**
Účel aplikace **25**
Původní účel aplikace TEN-T – zánášecí rozhraní (kodér) 25
Rozšíření aplikace TEN-T o přístup pro ŘSD 27
Transformace aplikace TEN-T – vznik aplikace CEBASS 29
Zázemí aplikace **30**
Použité technologie 30
Architektura aplikace 32
5. Testování aplikace **38**
Postup testování **38**
Použité nástroje **39**
Testování - Stránka s přihlašováním **39**
SQL Injection 39
LDAP injection 41
Implementace ochrany LDAP před útokem hrubou silou 42
Využití nástrojů OWASP 43
Správa relace (sezení uživatele), rozhraní pro výběr subsystému **44**
Implementace oprav 45
Rozhraní vkládající a zobrazující data **48**
Implementace opravy 50
Zabezpečení fotografií **52**
Implementace opravy 53
6. Závěr **55**
7. Používané termíny **57**
8. Seznam použitých zdrojů **59**
9. Seznam obrázků **64**
10. Seznam tabulek **66**

1. Úvod

Obecně lze říci, že pokud chceme klást velkou pozornost na bezpečnost, musíme provádět pravidelné kontroly a audity, s postupem času a výskytem dalších bezpečnostních hrozeb jsme nuceni přehodnocovat dřívější závěry, upravovat normy a přizpůsobovat metodiky. Tento fakt je široce aplikovatelný napříč různými odvětvími.

Aby byla tato kontrola bezpečnosti dodržována, nezřídka nám různé audity a testování diktuje přímo legislativa, a to jak při schvalování systému či díla do provozu, například formou kolaudace, tak pravidelnou kontrolou po daných intervalech v průběhu používání díla.

A to je přesně případ bezpečnostní inspekce na sítí TEN-T, která je v České republice uzákoněna od roku 2011, kdy vešla do působnosti vyhláška č 317/2011 Sb. (1). Tato vyhláška definuje povinnost provádět pravidelné bezpečnostní inspekce na silniční síti TEN-T a to jednou za 5 let. Fakt, že se tato vyhláška objevila v právním řádu ČR není náhodný, je to výsledek integrace směrnice 2008/96/EC Evropského parlamentu a rady (2).

Právě touto inspekcí byla pověřena ČVUT fakulta Dopravní a to od Ředitelství silnic a dálnic ČR, na základě výběrového řízení. V rámci tohoto projektu byl autor této práce společně s kolegou, jako pracovník správy sítě ČVUT FD, pověřen podporou v oblasti IT.

Zadáním bylo vytvoření jednotného webového formuláře a databáze pro evidenci zjištěných deficitů na silniční síti. Toto rozhraní mělo sloužit jako podpora pro jednotlivé auditory operátory, pro sjednocení formy a v neposlední řadě mělo pomoci v kontrole průběhu celé inspekce.

Postupem času a především díky narůstajícím požadavkům na funkčnost, se z původního, v zásadě jednoduchého webového formuláře, vyvinula plnohodnotná webová aplikace s oddělenou databázovou a aplikační vrstvou.

Jelikož aplikace našla zalíbení u zadavatele původní inspekce, tedy ŘSD ČR, které po zhlédnutí rozhraní a funkčnosti také zatoužilo po přístupu k této aplikaci a

jejím používání v rámci dalších inspekcí silniční sítě, bylo zapotřebí se zaměřit za zabezpečení této webové aplikace, aby byla použitelná v náročném prostředí a splňovala bezpečnostní standardy vzhledem k datům, které obsahuje. Díky faktu, že autor je hlavním programátorem právě zázemí aplikace, tzv. back-end[1] části, naskýtá se možnost podrobit tuto aplikaci testování na bezpečnost dle metodiky OWASP (3) a následně zaznamenat a rozpracovat popis implementace opravných opatření. Než však práce bude popisovat bezpečnost aplikace a následné opravy, představíme si organizaci OWASP, její metodiku a hrozby které zmiňuje.

Cíle práce

Cílem práce je analyzovat problematiku bezpečnosti webových aplikací nejen v obecné rovině, kdy za pomocí organizace OWASP a jejího žebříčku OWASP TOP 10 provedu rozbor hrozeb a to i s možnými návrhy opatření. Dále si práce klade za úkol zběžně představit aplikaci CEBASS, která vznikla na akademické půdě a našla využití ve společnosti ŘSD ČR, představím zázemí vzniku aplikace, ale především přiblížím architekturu aplikace a zmíním i její vývoj a pochopitelně účel. Po nezbytném představení aplikace provedu testování aplikace na bezpečnostní zranitelnosti, k nalezeným zranitelnostem se jako jeden z hlavních vývojářů vyjádřím, respektive udělám její rozbor a následně provedu sám implementaci opravy, která danou zranitelnost odstraní.

Bezpečnost aplikací je velmi široké téma s neustálým progresivním vývojem, předmětem tak jsou především ty nejzásadnější zranitelnosti, které v rámci testování budou odhaleny, pokud by mělo být testování hloubkové a v plném rozsahu a to včetně implementace řešení, práce by tak přesáhla svůj doporučený rozsah a to právě díky části implementace, která je náročná jak na vývoj, tak na otestování.

[1] Back-end – programátor zázemí aplikace, zpravidla práce takového programátora není na první pohled vidět, tento programátor se zabývá zpracováním dat na pozadí atp, jednoduše spravuje a vyvíjí to co je „pod kapotou“

Sekundárním cílem práce je také přiblížit téma bezpečnost webových aplikací z pohledu vývojáře, a to pro další začínající programátory, kteří začínají s minimem zkušeností stejně jako autor.

2. Úvod do testování bezpečnosti

Proč zrovna bezpečnost? Tento obor totiž v poslední době nabírá na velké důležitosti, často slýcháváme v mediích zprávy například o únicích osobních údajů z webových portálů, případně nedostupnosti nějaké služby či webu díky útokům, které na stránku jsou uskutečňovány. Tu a tam se také dozvíme obsah elektronické korespondence nějakého politika z důvodu špatné ochrany této korespondence a nedodržování systémových bezpečnostních procedur a opatření. Dále jsme informováni o faktu, že útočníci se zmocnili profilu politika nebo jiné významné osobnosti na sociálních sítích, dokonce se lze setkat i s jednotlivci, u kterých se toto vše stane během jednoho týdne.

Jaké ale máme nástroje k testování bezpečnosti? Jakým způsobem postupovat pokud se rozhodnu, že chci své dílo otestovat? Jaké aspekty přináší testování? Je to skutečně nutné?

Je zřejmé, že aplikace, která probíhá kontinuálním testováním v průběhu jejího vývoje je ve finále pro společnost, která ji vyvíjí levnější, než ta, která musí být následně po ukončení vývoje a nasazení do provozu přepracována z důvodu kritických bezpečnostních děr. Navíc nám testování v průběhu vývoje přináší ty výhody, že přijdeme na problémy dříve, než v produkčním nasazení, zvláště pokud se bavíme o testování funkční části aplikace, a to navíc vede ke zvýšení důvěryhodnosti naší aplikace před jejími uživateli v budoucnu.

Pro účely testování zabezpečení webových aplikací se osvědčilo takzvané **penetrační testování.** Toto testování má za úkol napodobovat počínání potencionálního útočníka na náš systém a to jak zevnitř, tak především zvenku. Penetrační testování jako takové však nestačí, výstupem takového testování je pouze odhalení potencionálních chyb v aplikaci, spolu s důsledky, které tyto chyby přinášejí a to jak pro provozovatele, tak pro aplikaci, tedy logicky po této části by měla následovat fáze oprav a záplatování aplikace. (4)

Nutno dodat, že penetrační testování nemá přesně psaná pravidla a postupy, úkolem testera je zkrátka dostat se do aplikace, případně jinak ji poškodit, v tomto směru by nás dodržování postupů a pravidel jen zdržovalo. Rozlišujeme také několik způsobů penetračního testování, takové, kdy útočník o inkriminované aplikaci neví vůbec nic, což se přibližuje profilu útočníka zvenčí, takovému testování říkáme **black box** testování. Také máme situaci, kdy naopak útočník, respektive tester, má kompletní znalost systému o jeho uspořádání, architektuře atp., tento stav se nazývá **white box** testování. (5)

Testování jako takové může probíhat automatizovaně, v tomto případě na základě nějakých scénářů, kdy se testují typické chyby v návrzích aplikací. Případné útoky provádí a vede jednotlivec, respektive skupina lidí, kteří tak hledají mezery v zabezpečení, případně se v reálném světě používá kombinace těchto dvou metod.

Na závěr je důležité zdůraznit, že penetrační testování není všespásné. U aplikace, která prošla testováním bez úhony, nelze tvrdit, že je bezpečná a neprolomitelná, jelikož se tento obor neustále vyvíjí a v průběhu času se objevují chyby v samotných bezpečnostních komponentech, například některé kryptografické algoritmy jsou časem překonány atp.

3. OWASP

O organizaci

Útoky na webové aplikace v poslední době rostou a především pojem bezpečnost na internetu je v poslední době velmi často zmiňována, například společně s ochranou osobních údajů, které webové aplikace často obsahují, například e-shopy o svých zákaznících, banky o klientech atp.

Nárůst útoků je patrně výsledkem toho, že internet jako takový k nim vybízí svou otevřeností a principem, že na něm jsme schopni nalézt vše, tedy i návod jak zaútočit či paralyzovat webovou aplikaci nebo stránku. Proto je v dnešní době myšleno, již od návrhu systému, na bezpečnost vyvíjených aplikací. Z pohledu vývojáře může však být náročné odhalit dané hrozby pro naši aplikaci, respektive si některé hrozby vůbec připustit nebo představit, zvlášť pokud se jedná o vývojáře s nedostatkem zkušeností.

Naštěstí však existují specializované portály, fóra a různé organizace, které vývojářům, ale i návrhářům, nebo chcete-li architektům aplikací, pomáhají ve vytváření webových aplikací tvorbou různých příruček formou na co si dát pozor. Tyto nástroje ale často nejsou určeny jen pro tyto IT profesionály, existují i nástroje, určené pro běžné klienty aplikací, kteří díky tomu jsou schopni dostat lepší pohled na aplikaci, kterou používají, aniž by byli závislí na slibech a přesvědčeních vývojářů.

The Open Web Application Security Project, ve zkratce OWASP, je nezisková organizace, zabývající se jak je z názvu patrné, bezpečností webových aplikací. Organizace byla založena v roce 2001 ve Spojených státech amerických jako komunitní organizace, založená za účelem zlepšení úrovně webových aplikací. Vše co OWASP nabízí (nástroje, dokumentace, fóra atp.) je poskytováno zdarma a navíc díky tomu, že se nejedná o žádnou komerční organizaci a faktem, že za organizací nestojí žádná jiná technologická společnost, je lépe zaručena objektivita a nezávislost v názorech, které organizace rozšiřuje. Všechny tyto prvky tvoří základní principy organizace. OWASP neposuzuje bezpečnost jen z pohledu aplikace, nýbrž zohledňuje i další faktory, například lidský a procesní faktor. (5)

OWASP tedy poskytuje vývojářům možnost využívat jejich služeb za pomoci různých dokumentací, nástrojů nebo příruček, případně poskytuje i návody. OWASP se dělí na několik tzv. projektů nebo částí, zmiňme ty nejdůležitější:

- OWASP Development Guide Project
  - Příručka sloužící vývojářům při tvorbě webových aplikací
  - V této příručce lze nalézt rady a tipy, jsou tam samotné ukázky kódů v různých programovacích jazycích

- OWASP Secure Coding Practices – Quick Reference Guide
  - Jedná se o příručku s obecnými radami a systémovými návrhy (ověřování uživatelů, logování, nastavení serveru atp)
  - Průvodce vývojáře na cestě k bezpečné aplikaci

- OWASP Application Security Verification Standard Project
  - Část, která je zaměřena právě na testování
  - Obsahuje seznam testovacích procedur

- OWASP Code Review Project
  - Opět část týkající se testování
  - Liší se načasováním testování
  - K testování dochází již v průběhu programování
  - Obsahuje přesně daný výčet způsobu kontroly a použité nástroje

- OWASP Testing Project
  - podobně jako code Review, aplikace je testována v průběhu vývoje
  - Také obsahuje přesně daný výčet způsobu kontroly a použité nástroje
  - Je zaměřeno na testování aplikace z pohledu uživatele, tzv. blackbox testování

- OWASP Application Security Verification Standard Project
  - Komplexní pohled na testování aplikace

- Zabývá se analýzou zdrojového kódu

Projektů, pod kterými je organizace podepsána je nespočetné množství, jen v rychlosti zmiňme ještě například OWASP WebGoat nebo OWASP Broken Web Application Project. Jedná se o projekt záměrně „špatně“ navržených aplikací, na kterých si uživatel může „osahat“ povahu a následky možných chyb v aplikaci. (6) Když už zmiňuji vzdělávací projekty, nesmím opomenout OWASP Education Project, kde nalezneme nepřeberné množství školících a prezentačních materiálů o dané problematice.

Organizace má ve světě na 200 poboček a můžeme být hrdi na fakt, že i Česká republika participuje na tomto projektu jednou pobočkou, nakonec OWASP je komunitní organizace, tedy každý může přispívat. Dokonce otevřenost této organizace je taková, že si můžete založit i pobočku vlastní. (7)



Obrázek 1 - Logo organizace OWASP (Zdroj: 3)

Mezi významné počiny, které má organizace OWASP na svědomí je uznávaný žebříček **OWASP TOP 10**, jehož obsahem je seznam 10 nejvážnějších hrozeb z pohledu webových aplikací, a to nejen to, poskytuje i rady jak se těmto hrozbám vyhnout.

OWASP TOP 10

Nyní si pojďme rozebrat žebříček nejvážnějších hrozeb. Nejaktuálnější seznam je z roku 2013 a v kontrastu s minulým žebříčkem organizací OWASP, zveřejněným v roce 2010, zde nalezneme podobné, často i stejné hrozby a stejně tak rozšiřuje jednu kategorii. Dále, mimo to, je žebříček rozšířen o zcela novou kategorii, zabývající se bezpečností komponent, ne že by problémy v rámci této kategorie nebyly dříve brány na zřetel, ale hrozby spadající pod tuto kategorii byly pouze okrajově zahrnuty v kategorii „nezabezpečená konfigurace“ (8), což bylo nepřehledné a dostatečně to nevyjadřovalo důležitost hrozeb, které tato kategorie zahrnuje.

OWASP Top 10 – 2010 (předešlé)	OWASP Top 10 – 2013 (nové)
A1 – Injektování	A1 – Injektování
A3 – Chybná autentizace a správa relace	A2 – Chybná autentizace a správa relace
A2 – Cross-Site Scripting (XSS)	A3 – Cross-Site Scripting (XSS)
A4 – Nezabezpečený přímý odkaz na objekt	A4 – Nezabezpečený přímý odkaz na objekt
A6 – Nezabezpečená konfigurace	A5 – Nezabezpečená konfigurace
A7 – Nechráněné kryptografické ukládání – sloučeno s A9 →	A6 – Expozice citlivých dat
A8 – Selhání v omezení URL přístupu – rozšířeno do →	A7 – Chyby v řízení úrovní přístupů
A5 – Cross-Site Request Forgery (CSRF)	A8 – Cross-Site Request Forgery (CSRF)
<pohřbeno v A6: Nezabezpečená konfigurace>	A9 – Použití známých zranitelných komponent
A10 – Neošetřené přesměrování a předávání	A10 – Neošetřené přesměrování a předávání
A9 – Nedostatečná ochrana transportní vrstvy	Sloučeno s 2010-A7 do nového 2013-A6

Tabulka 1 - Přehled kategorií 2010 a 2013 (Zdroj: 9)

A1 - Injektování

Injektování se dá popsat jako zneužití aplikace, respektive její části, k tomu, abychom z ní dostali informace, které při standardním zacházení nevrací. Typicky, například při nesprávné konfiguraci rozhraní, při komunikaci s databází SQL, případně při ověřování uživatelů LDAP. Jelikož je to opravdu velká hrozba, podle OWASPu dokonce největší, proberme si ji detailněji.

Tuto hrozbu si představme na konkrétním příkladu. Mějme například databázi uživatelů, uloženou v SQL serveru, tato tabulka má následující strukturu a záznamy:

ID	Jméno	Přijmení	Username	Heslo
1	Jan	Novák	Novakj1	MojeHeslo.
2	Martin	Nový	Novym2	DalsiHeslo.
3	Petr	Novotný	NovotnyP3	TretiHeslo.

Tabulka 2 - Tabulka uživatelů (Zdroj: Autor)

Dále předpokládejme, že aplikace používá na titulní stránce ověření, kde podle username si vybere z databáze konkrétního uživatele, všechny jeho atributy a heslo porovná s tím, které je v DB (pro ilustraci heslo ukládáme a v aplikaci jej zpracováváme čistě v textové podobě). Tedy náš dotaz použitý v aplikaci při komunikaci s DB je triviální a bude vypadat jako:

```
SELECT * FROM uzivatele WHERE username = $promenna
```
[2]

Dále předpokládejme, že aplikace pro předávání informací z formulářů používá metodu GET. Tedy hlavička po odeslání přihlašovacího formuláře bude vypadat takto:

```
mojeNovaApka.cz/prihlaseni.php?username=Novakj1&heslo=MojeHeslo.
```

[2] Pro ilustraci používám zjednodušený pseudojazyk vycházející z jazyka PHP (10)

Dotaz včetně „nabití“ proměnné $promenna bude vypadat takto:

```
$promenna = $_GET[¨username¨]; [3]
```

Proměnná $promenna má hodnotu „Novakj1“.

```
SELECT * FROM uzivatele WHERE username = Novakj1
```

Tento dotaz vrátí výsledek, ve kterém nalezneme atributy, které k záznamu patří dle tabulky. Než však připustíme uživatele do aplikace, ověříme, zda heslo, opět uložené v proměnné GET, souhlasí s heslem v DB, to však pro náš ilustrativní příklad není důležité.

Nicméně co když uživatel, nebo spíše útočník do proměnné napíše něco jiného? Pochopitelně záleží na tom co, ale pokud máme takovouto stavbu aplikace, která komunikuje s databází tímto způsobem, dáváme útočníkovi možnost z ní dostat informace, aniž by znal jakékoliv jiné údaje, jako je adresa SQL serveru, uživatelské jméno a username. Jednoduše upraví naši proměnnou $promenna tím, čím potřebuje. Tedy řekněme, že hlavičku GET upraví následovně:

```
mojeNovaApka.cz/prihlaseni.php?username=nevim OR
1&heslo=nezajem[4]
```

Tedy $promenna se nabije hodnotou „nevim OR 1“.

```
SELECT * FROM uzivatele WHERE username = nevim OR 1[5]
```

Výsledek tohoto dotazu je výpis celé databáze s uživateli, sice uživatel „nevim“ v naší databázi neexistuje, nicméně útočník injektoval náš dotaz pomocí spojení OR 1, tedy databáze sice první podmínce nevyhoví, ale druhé již ano, díky operátoru OR. Všimněme si, že útočník nepotřebuje znát ani strukturu databáze, respektive tabulky, stačí mu pouhé základy jazyka SQL.

---

[3] Proměnná $_GET v PHP obsahuje informace obsažené v hlavičce dokumentu v datovém typu pole, jednotlivé atributy se oddělují znakem „&“.

[4] Zjednodušené zobrazení, prohlížeč by mezery nahradil zástupnými znaky

[5] Spojení WHERE 1, má na svědomí v jazyce SQL úplný výpis databáze

Obrana

Bránit se proti této hrozbě v dnešní době lze již poměrně lehce, je však nutno přijmout nejlépe několik opatření. Obecně platí, že pokud chceme této hrozbě zamezit, musíme oddělit nedůvěryhodná data od používaných dotazů a příkazů. (11)

V první řadě platí, že vždy bychom měli ověřovat vstupy od uživatelů, nejlépe dynamicky, tedy ihned, typicky za pomocí jazyka JavaScript, jestliže tedy máme jasně danou syntaxi vstupní hodnoty, v našem případě username ve tvaru:

```
celePrijmeni+prvniPismenoJmena+ID
```

Pak pomocí JavaScriptu a regulárních výrazů jsme schopni kontrolovat daný vstup od uživatele a uživateli v reálném čase zamezit odeslání formuláře, nicméně podobná kontrola by měla být i v aplikaci, nebo chcete-li skriptu, zpracovávajícím vlastní proceduru, v našem případě spouští daný dotaz.

To samozřejmě problém jako takový neřeší, ale zabraňuje to částečně tomu, aby běžný uživatel omylem nedostal informace, které nemá.

Další věcí, která riziko injektování eliminuje, je používání takových rozhraní, tedy API, která jsou parametrizovaná, ve světě PHP tuto možnost, oproti knihovně *mysqli,* nabízí procedura pro komunikaci s DB **PDO**. Pokud však takové rozhraní neexistuje, OWASP nabízí své vlastní OWASP ESAPI (12).

A2 – Chybná autentizace a správa relace

V zásadě se dá říci, že v tomto případě aplikace nesprávně provede autentizaci uživatele při přihlášení, případně aplikace nesprávně spravuje sezení uživatele. Pod pojmem autentizace rozumíme ověření identity uživatele, tedy zdali uživatel má oprávnění k naší aplikaci, nachází se v databázi uživatelů a heslo sedí k tomuto uživateli. Následně je provedena autorizace uživatele, což bychom si mohli představit jako udělení jednotlivých práv v aplikaci.

Konkrétně tedy máme na mysli tu část, kdy potencionálním útočníkům zjednodušujeme práci tím, že důvěrné autentizační údaje ukládáme například v čistě textové podobě, případně pokud máme špatně nastavenou proceduru pro změnu hesla v případě jeho zapomenutí.

Dále zde nalezneme případy, které se týkají špatného nakládání s ID sezení, které umožňuje jeho „uhádnutí“, případně ukradnutí sezení, pomocí například přesměrování našeho uživatele na jiný web.

Pojďme si opět vše předvést na příkladu:

Mějme opět web s aplikací, s názvem mojeNovaApka.cz, zabývající se například prodejem elektra, předpokládejme, že uživatel se přihlásí do aplikace, nicméně ID sezení si ukládáme do proměnné GET, tedy ID sezení lze vyčíst z adresy:

```
mojeNovaApka.cz/index.php?prihlaseni=ok&idSezeni=11122334
```

Dále v našem scénáři si uživatel vyhlédne spotřebič, který se mu líbí, konkrétně televizor, adresní řádek pak bude mít podobu

```
mojeNovaApka.cz/index.php?prihlaseni=ok&idSezeni=11122334&ukazZbozi=tvSony
```

Uživatel se teď rozhodne tento odkaz poslat přátelům s prosbou o radu. Netuší však vůbec, že neposílá pouze odkaz na TV Sony, ale také údaj, který nese ID jeho sezení. Dá se tak předpokládat, že na základě tohoto ID nám pak mojeNovaApka.cz přidělí sezení původního uživatele, tedy například dostanu se do historie objednávek, nebo jsem schopen vidět údaje kreditní karty atp.

Obrana

V tomto případě představuje informovanost vývojáře tu nejlepší obranu, OWASP tak poskytuje v podobě manuálů a přehledů bezpečnostních rizik vývojářům návody, kde zmiňuje požadavky na správu relace a na postupy autentizace. Tyto sepsané standardy mají název Application Security Verification Standard. (13), případně OWASP poskytuje sadu ESAPI Authenticator and User APIs (14) s demonstrativními příklady, které by vývojáři měli použít.

V našem konkrétním výše uvedeném příkladu, bychom ID sezení ukládali v proměnné $__SESSION, a to v zahešované podobě, dále si aktuálnost sezení, respektive ověřování, zdali session z tohoto počítače skutečně patří tomu, kdo o ni teď žádá bychom měli kontrolovat, například ukládáním otisku prohlížeče nebo IP adresy (pozor však na mobilní přípojky a plovoucí IP adresy) do databáze.

A3 – Cross-Site Scripting (XSS)

V zásadě se jedná o podobný druh útoku jako u SQL injection, jednoduše řečeno útočník zneužije náš nezabezpečený vstup k tomu, aby docílil například změny stránky, případně si tak odešle citlivé údaje na svůj web. K tomu, aby se mu tato věc podařila, využije skriptování, podstrčí jednoduše naší aplikaci svůj JavaScriptový[6] kód. Velmi často je tento pojem zmiňován s phisingovými[7] útoky.

Existuje několik typů těchto útoků, pojďme si ukázat jeden velmi častý, který je označován jako persistent, tedy trvalý. Trvalý díky tomu, že se jedná o trvalou modifikaci nebo úpravu postižené stránky.

Mějme aplikaci mojeNovaApka.cz, kde je její obsah generován pomocí databáze, například opět předpokládejme e-shop s elektronikou. Pod každou položkou pak prostor pro vkládání dotazů či komentářů, které se pak následně zobrazí každému dalšímu zákazníkovi. Při neošetřeném vstupu pak útočník jednoduše do komentáře vloží svůj kód, například:

---

[6] JavaScript – univerzální skriptovací jazyk, používaný na WWW, skript je součástí stránky a provádí se tak až po jejím vygenerování (15)

[7] Phishing - Útok postavený na důvěře uživatele (stránka či e-mail se tváří věrohodně, je nerozeznatelná od oficiální) za účelem získání citlivých údajů (16)

```
<script>
document.getElementById("dostupnost").innerHTML =
"Vyprodáno";
alert(„Zboží vyprodáno“);
</script>
```

Následující skript přepíše hodnotu v atributu dostupnost na „vyprodáno“ a to bez ohledu na to, jaká informace se skutečně nachází v databázi, díky tomu, že JavaScript přepisuje již vygenerovanou stránku, dojde k přepsání původní hodnoty, zároveň uživateli vyskočí i okno se stejnou informací. Útočník tak může naprosto paralyzovat naše prodeje a to se jedná v podstatě ještě o „drobný“ žert, pomocí JavaScriptu, lze měnit i adresy odkazů atp.

Obrana

Podobně jako u SQL injection, je důležité ověřovat vstupy a tzv. escapovat data, která nejsou důvěryhodná, OWASP poskytuje příručku OWASP XSS Prevention Cheat Sheet (17).

U našeho příkladu můžeme například pro validaci dat, která „tiskneme“ na obrazovku použít v jazyce PHP deklarovanou funkci *htmlspecialchars()*[8].

A4–Nezabezpečený přímý odkaz na objekt

Tuto hrozbu si můžeme představit poměrně jednoduše. Je dost podobná předešlé, v podstatě se dá říci, že opět nám chybí kontrola vstupů a dělení na data důvěryhodná a nedůvěryhodná. Obecně tuto hrozbu si popišme jako možnost dostat se na stránku k údajům, které nenáleží mé osobě, a to pomocí přímého přístupu, například editací, tedy vstupem v adresním řádku.

Lépe to lze pochopit na příkladu s naším internetovým obchodem s elektronikou. Mějme stránku obsahující údaje o adrese zákazníka, tyto údaje se vyvolají pomocí

[8] Htmlspecialchars() – funkce převádějící některé html elementy na zástupné jiné znaky, tak že prohlížeč je nezpracovává a uživateli se tak objeví čistě jako text. (18)

funkce, která jako argument pro přiřazení těchto údajů používá údaje z adresní řádky, tedy proměnnou $_GET, kde nalezneme ID adresy. Mějme tak tuto podobu adresního řádku:

```
mojeNovaApka.cz/detailAdresa.php?adresa=222333
```

Nicméně pokud provedeme ruční změnu a ID zákazníka přepíšeme, rázem dostaneme údaje o jiném zákazníkovi v případě špatně zabezpečené aplikace.

Obrana

V zásadě je třeba kontrolovat, zda uživatel, který žádá o dané informace k nim má přístup, respektive oprávnění. Ideální je používat nepřímé odkazy na objekty, které jsou vygenerovány pro každého uživatele, případně sezení zvlášť. Typicky pak uživatel může mít v databázi vlastní tabulku, kde budou uvedeny vztahy pro převedení nepřímého odkazu na skutečný zdrojový, zároveň také fakt existence tohoto vztahu v této tabulce nám říká, že tento uživatel je oprávněn tyto údaje vidět.

Na konkrétním příkladu si tedy představme možné řešení, pro ilustraci přidejme představu, že přihlášený uživatel má v systému zaneseno více adres. Pro její zobrazení se opět používá funkce, která jako argument používá údaje z adresní řádky. Tabulka vztahů mezi zdrojem a nepřímým ID by pak vypadala takto, zdroj zde je cizí klíč z tabulky obsahující adresy:

id	zdroj
1	222333
2	222334
3	222335

Tabulka 3 - Příklad tabulky vztahů při nepřímé referenci (Zdroj: Autor)

Adresní řádek by pak byl v této podobě:

```
mojeNovaApka.cz/detailAdresa.php?adresa=1
```

Aplikace by pak provedla dotaz vůči databázi, který zdroj se nachází pod ID 1, následně by v tabulce adres nalezla ID zdroje a z tabulky adres by ukázala příslušné údaje. To samé i pro 2 a 3, v případě jiného čísla, které není přítomno v této tabulce, však napíše chybu a uživatele dále nepustí. Náš problém je tak vyřešen.

A5– Nezabezpečená konfigurace

Další z kategorie hrozeb, které OWASP má ve své TOP 10, jedná se o poměrně velkou kategorii, tedy její popis nastíním obecně.

Kategorie se, jak je z názvu patrné, zaobírá bezpečnostní otázkou konfigurací, které jsou používány pro provoz aplikace, typicky tak mluvíme o nastavení operačního systému, který aplikaci hostuje. Patří sem i aplikační software, který aplikace potřebuje ke své funkci, například webový server, moduly pro zpracování PHP (či jiných) skriptů, databázový server atp.

Obrana

Obecná rada je používat vždy co nejaktuálnější software, tedy udržovat i OS aktualizovaný, průběžně i provádět aktualizace balíčku softwaru, který je na serveru používán. Dále je velmi důležité věnovat úsilí a čas samotné konfiguraci serveru a jeho částí, především správně nastavit vzdálený přístup k serveru jako takovému, nastavit správně konfigurační soubory frameworků, které jsou používané, včetně vypnutí a zapnutí, například chybových hlášek, které jsou sice užitečné při vývoji aplikace, protože nám říkají, kde přesně se nalézá chyba, zároveň však odhalují potencionálnímu útočníkovi zdrojové kódy nebo i celé dotazy, které používáme k databázi.

A6 – Expozice citlivých dat

Toto se týká dat, která ukládáme v aplikaci, respektive v databázi a mají povahu citlivých údajů, tedy řadíme sem takové údaje, jakými jsou čísla bankovních karet, zdravotní záznamy, osobní údaje, ale také hesla.

Tyto údaje potřebují z principu zvláštní ochranu a také bychom s nimi měli i jinak zacházet. Tím myslíme zálohování, komunikaci s databází, přenosy skrze síť(e).

Je tedy důležité náležitě tato data chránit, například krytováním a samozřejmě používat i dostatečně silné algoritmy krytování, jelikož v dnešní době jsou některé algoritmy již překonány a jejich efekt je naprosto stejný jako kdyby data byla uložena v čistě textové podobě. Dále je třeba v rámci přenosu, či jakéhokoliv nakládání s daty, používat takové nástroje, které poskytují dostatečně bezpečnou cestu.

Obrana

Obrana je dle předchozího odstavce zřejmá, konkrétně se doporučuje data šifrovat, používat dostatečně silné klíče, nedržet citlivá data tzv. zbytečně (přináší jen starosti a odpovědnost). Ideální je používat takové algoritmy pro ukládání hesel, které k tomu jsou určeny.

A7 – Chyby v řízení úrovní přístupu

Tato chyba se dá přiblížit tak, že aplikace nedostatečně ověřuje roli uživatele nebo ji vůbec nerozlišuje.

Mějme například stránku, kde se nám může přihlásit uživatel „pepa“ a uživatel „admin“. Po přihlášení jej webový formulář odkáže na jinou stránku v případě „pepa“ by tato adresa byla tato:

```
mojeNovaApka.cz/prehledClanku.php
```

Pro „admin“ by zase adresní řádek vypadal takto

```
mojeNovaApka.cz/prehledClanku_admin.php
```

Jen pro ilustraci si představme, že „admin“ je správce webu a může přidávat a mazat články na této stránce, na straně druhé je „pepa“ běžný uživatel, který má nárok po přihlášení tyto články číst.

Chyba v řízení úrovní přístupu v tomto případě by se tak mohla projevit za situace, kdy se „pepa“ přihlásí a úmyslně přepíše URL stránky v adresním řádku na to, které odpovídá pro admin, případně změní nějaký parametr nebo argument. Důležité pak je, jak se zachová aplikace? Pustí uživatele „pepa“ do administračního rozhraní nebo si před zobrazením stránky ověří jeho práva, případně roli, kterou v systému má?

Obrana

Nutno dodat, že naštěstí proti těmto útokům existuje poměrně jednoduchá obrana a tou je integrace autorizačního modulu do aplikace, tedy představme si, že naši aktéři mají přidělenou každý svoji roli „admin“ a „čtenář“. Dále si napíšeme funkci, která bude tyto role ověřovat s právě přihlášeným uživatelem, tedy řekne nám při každém načtení stránky „pepa“ má roli „čtenář“. Následně, vhodným systémem přidělíme jednotlivým stránkám oprávnění, co jaká role může zobrazovat a vždy, než nám je ukázán obsah požadované stránky, se provede autorizace požadavku. V případě negativního výsledku, tedy kdy je zamezeno zobrazení stránky, je na místě tuto událost náležitě logovat.

A8 - Cross-Site Request Forgery (CSRF)

Tento druh útoku je zdánlivě podobný XSS, nicméně jedná se o dva různé útoky, byť jistou podobnost u nich určitě lze spatřit.

Cross-site Reques Forgery je založeno na tom, že potencionální útočník „podstrčí“ uživateli našemu systému, který je přihlášen, obsah, který pak naše aplikace zpracuje ve prospěch útočníka a to často i bez toho, aniž by náš uživatel o tom věděl. Částečně se také dá říci, že je tento typ útoku postaven na důvěře. Pojďme si pro lepší ilustraci tento útok rozebrat na příkladu.

Mějme opět zadanou naši fiktivní webovou aplikaci, například náš obligátní e-shop s elektronikou. Na tomto e-shopu předpokládejme, že je možnost platit za zboží z předem předvolené platební karty, nicméně systém teoreticky dovoluje platit i za jiné objednávky na základě objednacího čísla, které ve standardním stavu je předvoleno na konkrétní objednávku a s požadavkem se posílá pomocí metody GET, tedy nalezneme jej v adresním řádku. Pro spuštění platby by náš adresní řádek mohl vypadat nějak takto:

```
mojeNovaApka.cz/platbaKartou.php?objednavka=111222
```

Následující požadavek tak vykoná platbu z naší uložené karty a zaplatí tím objednávku s číslem 111222.

CSRF je založeno na tom, že v rámci našeho aktivního sezení, kdy jsme zkrátka přihlášeni k e-shopu, je naše pozornost odlákána na jinou stránku, kde se nám zobrazí nějaký obsah, například captcha obrázek, reklama, zkrátka cokoliv. Náš

útočník tak vyláká nevinného zákazníka na svou stránku, kde mu zobrazí následující kód

```
<img src="http://mojeNovaApka.cz/platbaKartou.php?objednavka=222333
"/>
```

Následující „obrázek“ je pak prohlížečem zpracován a v případě aktivního sezení s naším e-shopem je v podstatě nevědomky odeslána platba z naší karty na cizí objednávku.

Na první pohled by se však mohlo zdát, že aplikace je háklivá pouze na formuláře nebo akce pracující s metodou GET, nicméně není tomu tak, v případě že používáme metodu POST i tak ke zneužití může dojít, byť způsob oproti GET již není tak triviální, jelikož útočník na své stránce musí použít kombinaci JavaScriptu s HTML formulářem.

Zajímavostí v souvislosti s touto hrozbou je, že i samotný Google tuto chybu obsahoval, byť její následky rozhodně nebyly tak drastické a k odstranění došlo již před dlouhými 8 lety. Jednalo se o chybu, kdy bylo možné pomocí CSFR infiltrace změnit výchozí jazyk vyhledávače, tedy chyba co nijak neublíží, ale příjemná jistě není. (19)

Obrana

I v tomto případě obrana proti těmto útokům není tak složitá, jak by se na první pohled mohlo zdát. Například jedno z možných řešení je generovat buď k sezení uživatele nebo ke každému formuláři nějaké náhodné ID, a to ukládat v HTML elementu input s atributem hidden a zároveň si ID uložit na server jako to „správné“, nedávno vygenerované, pak jednoduše porovnat tyto hodnoty a útočníkova stránka je bez šance, pochopitelně za předpokladu, že se mu nějak nepodaří dostat se k této hodnotě, protože by byla nepatřičně chráněna nebo za předpokladu, že se uživatel nachází spolu s útočníkem na stejné bezdrátové síti a útočník je tak schopen sledovat naši síťovou komunikaci.

Další obranou, nebo spíše prevencí, je dostatečná informovanost uživatele. Konkrétně by měl být v rámci nějaké instruktáže nebo schválení podmínek k používání naší aplikace poučen o tom, že v případě přihlášení a aktivní relace

v naší aplikaci je nebezpečné prohlížet jiné nedůvěryhodné weby. To je ale samozřejmě spíše utopistická představa.

Naopak k nepotřebě jsou ochrany typu ověřování otisku prohlížeče nebo kontroly IP adresy, jelikož z tohoto pohledu je požadavek vůči aplikaci autorizovaný, iniciuje jej totiž počítač uživatele, ne útočníka.

Další účinnou ochranou je v případě nějaké citlivější operace po uživateli žádat další ověření, ať už jednoduše pomocí CAPTCHA[9] obrázku, tak pomocí znovu přihlášení.

A9 – Použití známých zranitelných komponent

Hrozba, která často odráží velikost lenosti administrátora serveru, případně vývojáře. Je totiž založena na používání zastaralých a již překonaných technologiích zabezpečení.

Na jejich obranu je však nutno dodat, že ne vždy jsou administrátoři dobře informováni o existující chybě v nějaké komponentě, nakonec je to tak i dobře, jelikož jak komerční, tak i open source software, často řeší při nalezení určité bezpečnostní díry dilema, zda jej okamžitě zveřejnit, pochopitelně společně s opravou nebo to „držet pod pokličkou“, zkušenost totiž říká, že na tyto zprávy zpravidla rychleji zareagují potencionální útočníci, než administrátoři serverů a vývojáři.

Důležitost této kategorie navíc dokazuje fakt, že existuje řada nástrojů, které zanalyzují náš web a hledají zranitelné komponenty, což poskytuje především útočníkům výhodu v tom, že nějaký automatizovaný systém jim jen hodí seznam potencionálně napadnutelných webů a ti se pak pustí do díla.

[9] CAPTCHA – systém ověřování na přítomnost uživatele, určen převážně proti internetovým botům, jednoduchým vygenerováním řetězce, který je pak následně po uživateli požadován k opsání

Obrana

Obrana je zdánlivě jednoduchá, ale jak bylo nastíněno výše, často chybí informovanost a navíc především u open-source softwaru problém s verzováním některých modulů či knihoven, proto rada mít co nejaktuálnější softwarovou výbavu serveru je velmi lehce vyslovitelná, ale složitější již na splnění.

Tedy kromě klasického sledování aktuálnosti jednotlivých repozitářů se softwarem bychom měli i sledovat informace na veřejných fórech, na bezpečnostních konferencích, případně sledovat různé bulletiny, zabývající se zabezpečením, případně využívat co nejvíce vlastních knihoven, které si napíšeme sami a o možných problémech zpravidla víme.

A10 – Neošetřené přesměrování a předávání

Poslední, nicméně ne méně významnou kategorií, patřící pod OWASP TOP 10, je kategorie A10, zahrnující hrozby v kategorii nešetřené přesměrování a předávání.

Základním předpokladem pro existenci této hrozby v aplikaci je fakt, že používá pro přesměrování uživatele nějakou vnitřní funkci, která jako parametr používá metodu GET. S tímto se můžeme setkat v případech, kdy například potřebujeme logovat pohyb našeho uživatele nebo pro různá počítadla kliknutí atp. Další využití této funkce je v případě, že chceme na základě nějaké události uživatele přesměrovat na jiný web, respektive doménu. Naše volání takového přesměrování pak může vypadat následovně:

```
mojeNovaApka.cz/presmeruj.php?url=www.mojeStaraApka.cz
```

Je zřejmé, že v případě neošetřené samotné funkce, která provádí přesměrování je tento stav velmi lehce zmanipulovatelný útočníkem, který naší oběti může poslat odkaz mailem a v proměnné url použije svůj web například se škodlivým kódem.

```
<a href="mojeNovaApka.cz/presmeruj.php?url=badWeb.cz">Pro vtip dne klikni!</a>
```

Obrana

Obranou je funkce přesměrování používat co nejméně a v případě jejich použití platí to samé jako u ostatních věcí, prověřovat a ověřovat vstupy, v našem případě

hodnotu argumentu URL, například pomocí porovnání ze seznamem známých webů pomocí výpisů z SQL databáze. Případně používat jako parametr pouze mapovací hodnotu, která je následně přeložena na skutečný web.

OWASP Testing Guide

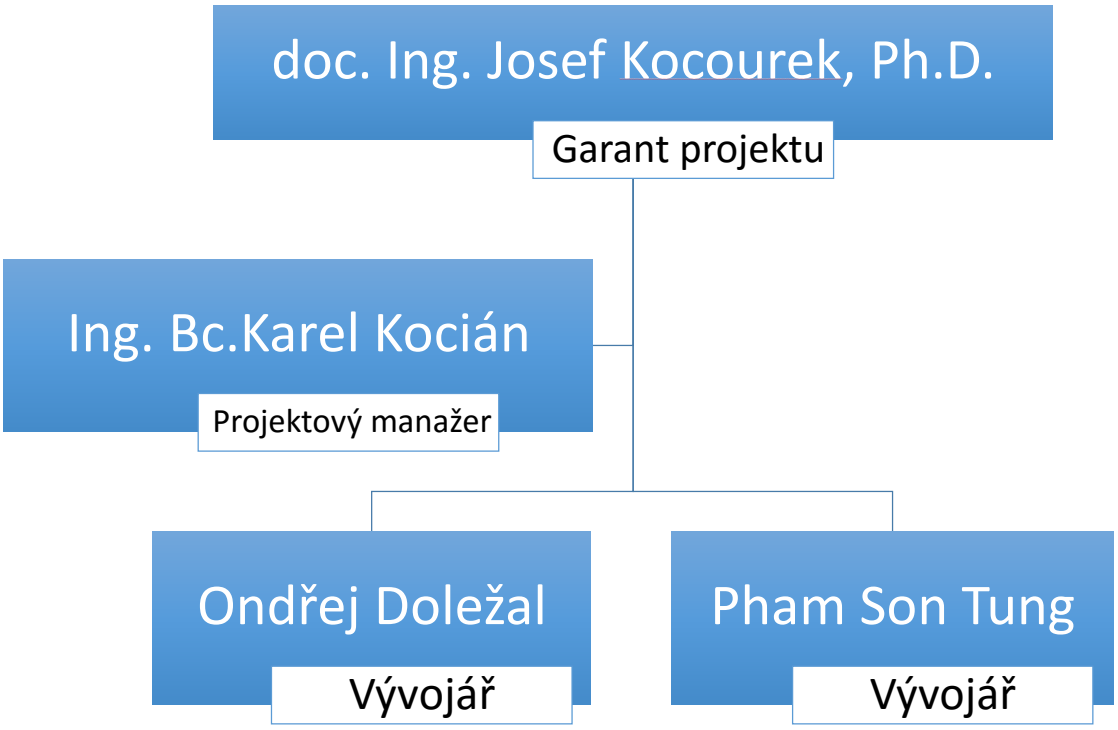
V souvislosti s organizaci OWASP a testováním bezpečnosti musíme zmínit příručku testera, tzv. OWASP Testing Guide v4 (20), ve které jsou zmíněny postupy při testování bezpečnosti a zmíněné doporučené nástroje pro testování. Nakonec, právě této příručky, se bude tato práce při testování držet a využívat ji.

Tato příručka je vyústěním práce projektu Owasp Testing project. Nezmiňuje pouze nástroje k testování ale i postupy. Klade důraz na fakt, že sice existuje nepřeberné množství zautomatizovaných pomůcek, které jsou určené k testování bezpečnosti, nicméně tyto nástroje slouží k zefektivní procesu testování, nelze se na ně spoléhat jako na nástroj, který řekne jak je aplikace bezpečná či nikoliv. Tyto nástroje jsou obecně napsané, neznají naši strukturu aplikace, další specifika, která mohou výsledek testování ovlivnit.

4. Aplikace CEBASS

Nyní pokročíme k samotné webové aplikaci s názvem CEBASS, tento název není náhodný je to zkratka spojení „Centrální evidence pro bezpečnostní analýzu silniční sítě“. Tato aplikace dále bude předmětem našeho testování v rámci bezpečnosti. Rád bych Vám v následujících kapitolách aplikaci představil, objasnil úlohy autorů v projektu CEBASS, vysvětlil účel aplikace, kde mimo jiné najdete několik ukázek a lehký nástin funkcí, které aplikace má. Pak popíši zázemí aplikace, především architekturu a technologie, které byly při jejím vytváření využity. Až pak samotná kapitola bude vyhrazena testování na bezpečnostní hrozby včetně implementace řešení.

Autoři



Obrázek 2 – Autoři aplikace (Zdroj: Autor)

Než tedy překročíme k samotnému popisování aplikace, ať už po funkční stránce nebo po stránce technické, sluší se představit autory této aplikace, respektive tým osob, bez kterých by nevznikla. Za projektem od samého začátku stál jako garant projektu **doc. Ing. Josef Kocourek, Ph.D.**, který zastřešoval projekt TEN-T obecně jako takový, dále **Ing. Bc. Karel Kocián**, který sloužil jako metodik pro

vyhodnocování deficitů na silniční síti díky svým dřívějším profesním zkušenostem v tomto oboru, zároveň však působil jako projektový manažer, respektive jako koordinátor vývojových prací na projektu CEBASS, je to právě ten člověk, kterým jsem, společně se svým kolegou, byl poptán v první řadě na vytvoření „jednoduchého“ webového formuláře pro účely ulehčení práce při zanášení deficitů.

A konečně pro účely této práce se dostáváme ke stěžejním osobám v projektu CEBASS a tím jsou programátoři, respektive vývojáři aplikace. Můj kolega, **Pham Son Tung**, který mimo jiné je také studentem VŠE FIS, kde právě píše bakalářskou práci na téma aplikace CEBASS, kde se zabývá vývojem především front-endu aplikace a představuje hlouběji funkce, které aplikace má. Z principu věci výše je můj kolega zodpovědný převážně za back-end aplikace, tedy na starosti měl převážně grafické zpracování dat, různé výstupy do jiných formátů, například XLS (Excel), ale jelikož naše práce nebyla striktně oddělena (např. návrh databáze jsme navrhovali spolu), najdeme jej podepsaného i v některých částech back-end aplikace, stejně tak jako mne ve front-endu (například výstup do PDF). Kolega disponuje pokročilou znalostí jazyku HTML, CSS, JavaScript, ale také znalostí PHP a SQL.

Moje úloha v projektu CEBASS, jak jsem již zmínil, byla pozice převážně back-end vývojáře a administrátora serveru. Můj výčet práce na aplikace tedy byl následující:

- Přihlašování a ověřování uživatelů (lokální, LDAP), definice rolí a správa relací, přiřazení aplikace s množiny CEBASS
- Komunikace s databází
- Model práv uživatelů
- Výstupy do PDF
- Rozhraní výpis deficitů (grafika i data) a vkládání reakcí
- Rozhraní na změnu hesla

Dále jako správce serveru jsem prováděl běžné činnosti jako:

- Instalace a nastavení LINUX serveru (základní nastavení, zabezpečení atp.)

- Instalace a správa SSL certifikátu
- Zálohování databáze
- Pravidelná aktualizace software na serveru

Jak jsem již zdůraznil, s kolegou Phamem jsme neměli striktně dané mantinely, jaká oblast je v jeho kompetenci a jaká ne, zkrátka co bylo potřeba, jeden z nás udělal a dále již také spravoval.

Velkým přínosem při vývoji této aplikace se pro nás stalo naše studium na VŠE, konkrétně absolvování předmětů 4IZ218 Databáze, kde jsme naplno využili poznatky z tohoto předmětu při návrhu databáze a znalost SQL jazyka, kterou používáme v podstatě denně, dále předmět 4IZ228 Tvorba webových stránek a aplikací, kde jako zajímavost mohu říci, že jsou použity základy při přihlašování uživatelů podobné těm, které jsem používal pro moji semestrální práci pro tento předmět a v neposlední řadě předmět 4SA313 Bezpečnosti informačních systémů, který mne díky své náplni natolik nadchnul, že jsem si jej vybral jako téma své práce, pomohl mi lépe pochopit princip asymetrického šifrování, SSL certifikátů a také jsem objevil nástroje pro testování bezpečnosti serveru jako takového.

Účel aplikace

Původní účel aplikace TEN-T – zánášecí rozhraní (kodér)

Účel už byl lehce nastíněn výše, aplikace původně vznikla jako webový formulář, kde bylo možné pomocí rolovacích seznamů a předem nastavených formulářových prvků zadávat požadované údaje o nalezeném deficitu, tyto deficity zanášeli operátoři (kodéři) proškoleni metodikem a to pomocí videa, které si pouštěli na monitoru a v případě nalezení deficitu sejmuli snímek obrazovky a vložili jej do aplikace, spolu s GPS souřadnicemi, které byly součástí videa. V rámci co největší automatizace byl mechanismus vkládání tohoto snímku obrazovky řešen pomocí elementu HTML5[10] canvas a JavaScriptu, operátorovi

[10] HTML5 – verze značkovacího jazyka HTML, jedná se o základní jazyk při tvorbě webu, tato verze přináší nové elementy a funkce (21)

tak stačilo jen udělat snímek a pomocí CTRL+V jej vložit do okna canvas a obrázek byl uložen.

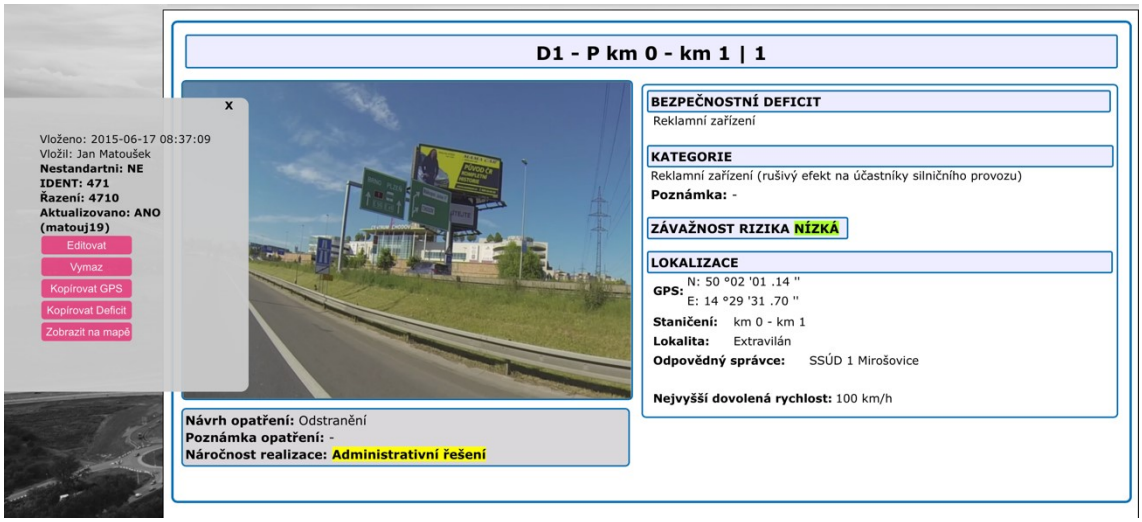
Tento webový formulář pak údaje vyplněné ukládal do MySQL databáze (později byla použita databáze MariaDB), už v této fázi aplikace rozlišovala jednotlivé uživatele a dokonce je ověřovala vůči LDAPu. Toto vkládací rozhraní umožnovalo jednoduché zanášení údajů, součástí byla i kontrola těchto údajů, zda je vše a řádně vyplněno a až poté bylo možné deficit uložit.



Obrázek 3 - Zanášecí rozhraní (Zdroj: Autor)

Nicméně v průběhu postupu projektu se začaly množit nároky na aplikace, především po stránce funkčnosti. Bylo tak do aplikace dopracováno několik věcí:

- Editace již vloženého záznamu
- Označení záznamu jako nestandardního (v případě nějaké nejasnosti tak operátor označil záznam, který by měl být zkontrolován koordinátorem)
- Prohlížení již vložených deficitů s pomocí různých filtrů (dle komunikace, staničení, příznaku nestandardní, dle operátora, kategorií deficitů atp.)
- Kopírování informací do nového deficitu
- Zobrazení deficitu na mapě



Obrázek 4 - Položka ve výpisu pro operátora (kodéra) (Zdroj: Autor)

Původně tímto měl vývoj na aplikace pro nás skončit, respektive poslední věcí měl být modul na export deficitů do PDF, jelikož jednotlivé deficity měly být předány ŘSD ČR v listinné podobě, rozdělené dle komunikací, tak jako v minulosti.

Nicméně po prezentaci této části aplikace na generálním ředitelství, kdy aplikace nebyla předmětem této prezentace nýbrž se jednalo o konzultaci týkající se několika deficitů z této schůzky pak vzešel požadavek připravit aplikaci také pro zaměstnance ŘSD ČR a její místní správy.

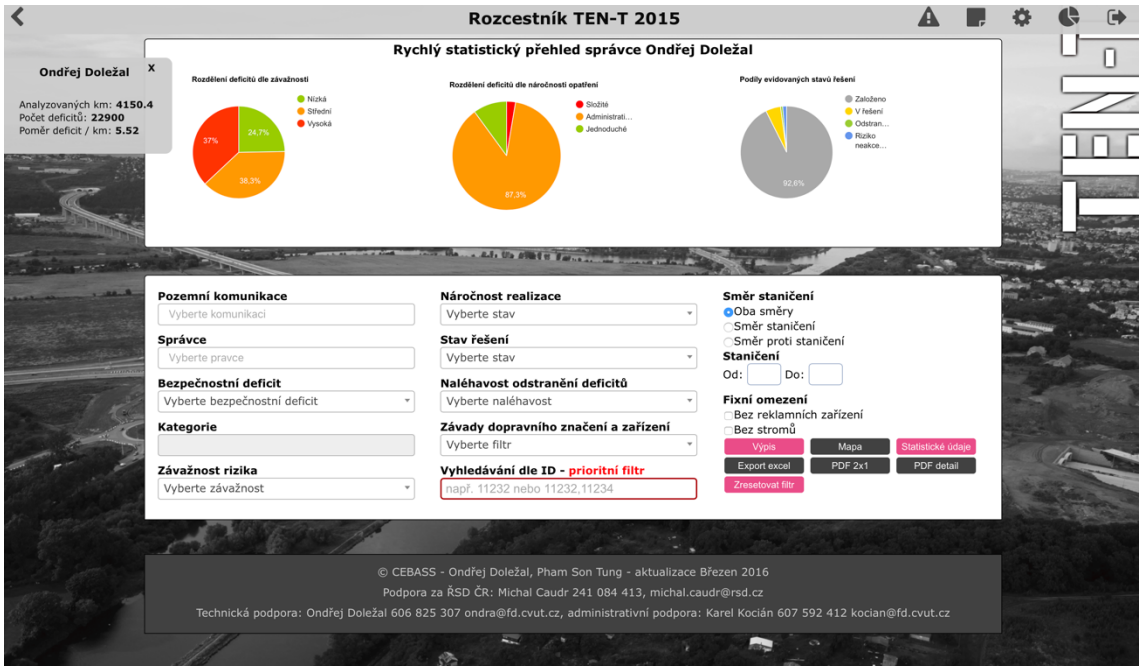
Rozšíření aplikace TEN-T o přístup pro ŘSD

Na základě této skutečnosti tak začaly práce na této části, bylo třeba přidat další uživatelské role a lokální ověřování uživatelů, bylo třeba tyto části od sebe i náležitě oddělit, části pro kodéra a pro ŘSD. Navíc přibyl i nový požadavek aby zaměstnanci ŘSD mohli na jednotlivé deficity reagovat.

Oblasti, které bylo třeba zapracovat:

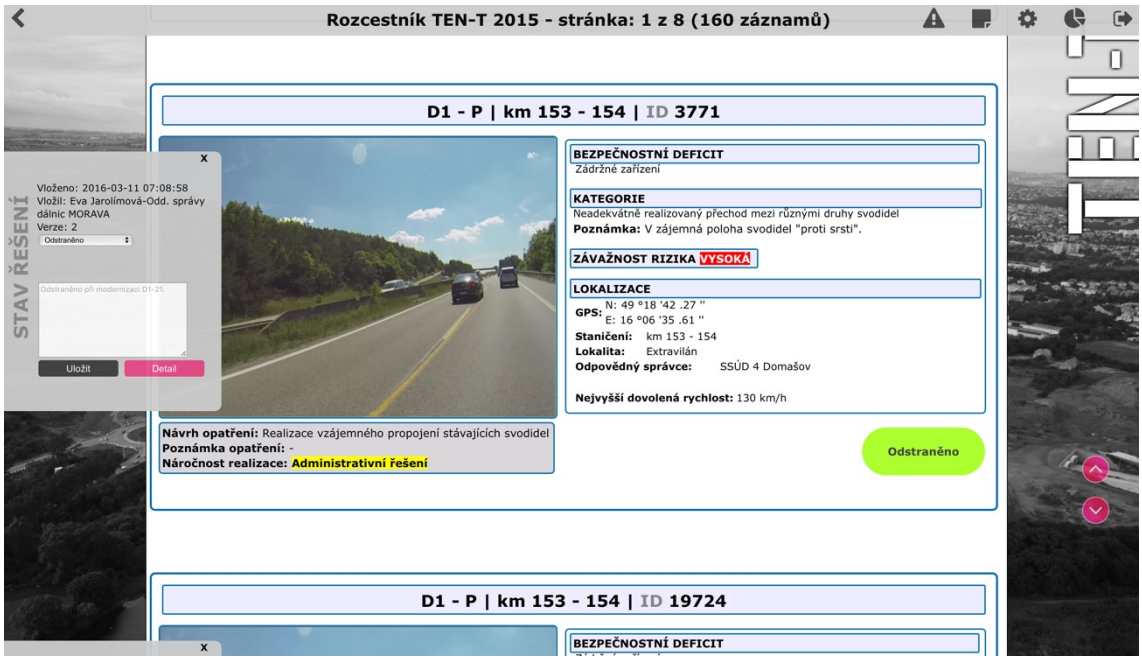
- Nové role:
    - ŘSD správce („nejnižší“ role) uživatel může reagovat pouze na ty deficity, které mu náleží (SSÚD Pravy, Mirošovice atp.)
    - ŘSD super-správce (nadřazený správci) jedná se o subjekt který pod sebou sdružuje jednotlivé správy (Závod Brno, Závod Praha) uživatel s touto rolí se může vyjadřovat ke všemu, co spadá správcům v jejich působnosti
    - ŘSD manažer (nejvyšší instance) – může reagovat na vše

- Grafické výstupy dat, grafické znázornění dat
- Pokročilejší filtrování
- Evidence průběhu stavu řešení tzv. WorkFlow
- Ověřování uživatelů vůči lokální databázi
- Možnost změny hesla
- Souhrnné rychlé grafy pro jednotlivé správce
- Historie posledně vložených stavu řešení



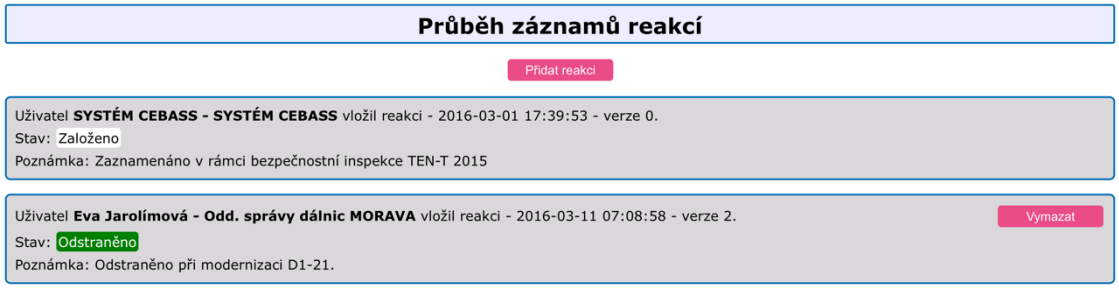
Obrázek 5 - Rozcestník pro uživatele ŘSD (Zdroj: Autor)

Na přiloženém snímku je vidět, že se dílo podařilo a byl vyvinut naprosto nový rozcestník s interaktivními funkcemi, například obsahuje dynamické vyhledávání a přiřazování hodnot na základě výběru jiné, konkrétně při výběru komunikace D1 se roleta správce zaktualizuje a zobrazí pouze ty správce, kteří mají na této komunikaci nějaký úsek zaevidovaný v databázi. Také byly dokončeny tzv. rychlé grafy, které pracují s daty v reálném čase, tyto grafy si může každý správce zobrazit i pro jiného správce a porovnat tak počet deficitů a stavy řešení.



Obrázek 6 - Rozhraní výpisu s oknem stav řešení (Zdroj: Autor)

Na dalším snímku vidíme rozhraní výpisu, kde se nachází část pro vkládání reakcí, tedy stav řešení. Ten se skládá z komentáře a z vybraného stavu řešení s tím, že je možné jich vkládat více za sebou, odlišují se tak verzí a tento „postup“ stavu řešení pak je uživatel schopen vidět v detailu daného deficitu, což si ukážeme na dalším snímku.



Obrázek 7 - Průběh záznamu reakcí (WorkFlow) (Zdroj: Autor)

Transformace aplikace TEN-T – vznik aplikace CEBASS

V průběhu používání původní aplikace TEN-T a stále se zvyšujících nároků bylo jasné, že časem se budou muset jednotlivé části (kódovací, prohlížecí a ŘSD) více oddělit, navíc přišla poptávka na provedení dalších inspekcí pouze s jinou tématikou, případně rozšíření inspekcí typu TEN-T o úseky nižších tříd (poptávka od jednotlivých správců), pochopitelně aby vše bylo přístupno pod jedním loginem, požadavek byl také na jednotnost ovládání. Tak byla původní

architektura aplikace a databáze předělána a uživatel, který má přiřazeno více podsystémů si tak po přihlášení může vybrat, který systém v rámci relace bude používat.

Vznikl tak systém CEBASS, tedy Centrální evidence pro bezpečnostní analýzu silniční sítě.

Při výběru požadovaného systému se uživateli přizpůsobí jeho relace, připojí odpovídající databáze a pohybuje se v kořenovém adresáři vybrané aplikace. Více o tomto stavu popíšu v architektuře aplikace.



Obrázek 8 - Výběr podsystému (Zdroj: Autor)

Zázemí aplikace

Každá webová aplikace má svá specifika a požadavky na výkon a funkcionalitu, od tohoto se pak mění požadavky na server jako takový, jak po HW stránce, tak po stránce SW, kdy je třeba ve spojitosti s požadavky na určitou funkci zvolit odpovídající programovací jazyk, respektive když mluvíme o webových aplikacích, mluvíme spíše o kombinaci programovacích jazyků.

Použité technologie

Pro aplikaci TEN-T byl od začátku zvolen programovací jazyk PHP a to ve strukturované formě, díky požadavkům na co nejvíce interaktivní prostředí byl

pro potřeby aplikace použit i skriptovací jazyk JavaScript, často s knihovnou jQuery[11], která přináší v mnoha ohledech výrazné zjednodušení práce pro vývojáře.

PHP a JavaScript jsou z principu naprosto rozdílné jazyky a to jak z pohledu syntaxe, tak z pohledu zpracování. JavaScript je zpracováván klientskou stranou, díky této vlastnosti také dokáže interaktivně reagovat na uživatele a je tak oblíben při vývoji webových aplikací. PHP je jazyk zpracováván čistě serverem a jeho zdrojové kódy, na rozdíl od JavaScriptu, tak nejsou vidět. Zdánlivou nevýhodou PHP je fakt, že server stránku zpracuje i zobrazí a pro její změnu na základě interakce uživatele je potřeba znova načtení stránky s novými parametry. Nicméně toto omezení se právě dá vyřešit propojením s JavaScriptem pomocí technologie AJAX, která umožnuje asynchronně zaktualizovat jen část stránky a zobrazit tak odpovídající obsah, stejně tak jako dokáže vzdáleně v reálném čase spouštět jiné skripty a dá se tak využít například pro ukládání obrázku, spouštění skriptů na práci s databází atp.

```
function historieReakci(){
    $.ajax({
        url: '/includes/historie.php',
        beforeSend: function(){
            $('#loader-icon').show();
            $("#obsahRSD").hide();
        },
        success: function(data){
            $('#loader-icon').hide();
            $("#obsahRSD").fadeOut('slow').empty();
            $("#obsahRSD").fadeIn('slow').append(data);
        },
        error: function(){}});

}
```

Obrázek 9 - Ukázka kombinace JS s AJAXem hledající historii reakcí k deficitu (Zdroj: Autor)

Výčet programového vybavení serveru:

- OS: Linux Ubuntu server 14.04.03 LTS 64bit
- Balíčky:
    - Apache2 (webový server v2.4.7) (22)

[11] jQuery – JavaSkriptová knihovna určená ke zjednodušení práce s JS (23)

- PHP (v5.5.9) + rozšíření pro LDAP a další knihovny (MySQL, mcrypt atp.) (24)
- MariaDB (openSource SQL databáze, fork MySQL projektu) (25)
- phpMyAdmin (grafické rozhraní pro přístup k DB) (26)
- htop (sledování využití prostředků serveru) (27)
- mPDF (knihovna psaná v jazyce PHP určená k sestavování a generování PDF dokumentů) (28)
- PHPExcel (knihovna psaná v jazyce PHP určená k sestavování a generování dokumentů ve formátů excel, tedy xls a xlsx) (29)

Hardwarové vybavení serveru

Server běží jako virtuální stroj za pomocí technologie společnosti VMware. Jedná se o server s procesorem Intel Xeon s osmi fyzickými jádry a technologií HyperThreading, tedy celkově je k dispozici 16 vláken, server je dále osazen 48GB operační paměti RAM, 4 plně duplexními zásuvkami typu ethernet s rychlostí až 1Gbps a hardwarovým řadičem RAID pracující v módu 5.

- Přiřazené prostředky:
  - 6 GB operační paměti RAM
  - 4 virtuální procesorová jádra
  - 1 přípojka ethernet plně duplexní mód s rychlostí 1Gbps
  - 250 GB prostoru z diskových oddílů

Architektura aplikace

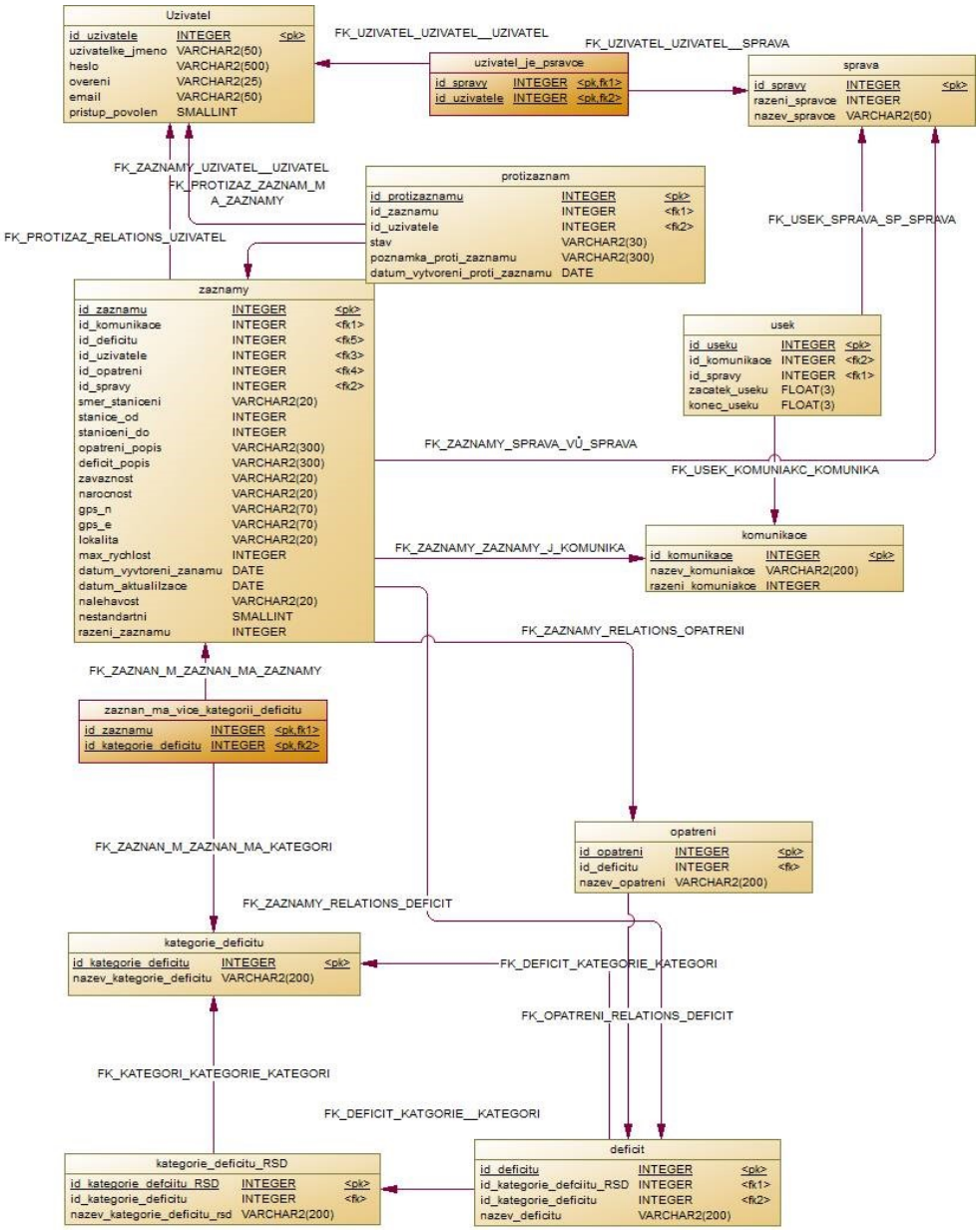
Architektura aplikace se díky postupnému zlepšování a měnícím se nárokům postupně také měnila, největší změnu však architektura zaznamenala při rozhodnutí na vytvoření univerzální centrální aplikace, která bude sdružovat různé systémy, tedy vznik aplikace CEBASS.

Záměrně uvádím jak původní architekturu, tak aktuální, jejíž představení považuji za nezbytné pro případné aplikace řešení bezpečnostních hrozeb. Na schématech je vidět progres aplikace, její posun a zároveň částečně tato schémata nastiňují rozsahy práce.

Původní aplikace TEN-T

Původně aplikace byla zaměřena pouze pro účely evidence deficitů na silniční síti TEN-T, tedy jak říká legislativa, v tomto případě se jedná pouze o dálnice, rychlostní silnice (nově od roku 2016 jen dálnice) a vybrané silnice prvních tříd. Z toho důvodu byly všechny části, včetně ověřování uživatelů, sloučeny v jednu aplikaci, která se jen dělila na základě role na rozhraní kodéra a na rozhraní prohlížení (později přibyla funkce přidávat reakce v tomto rozhraní).

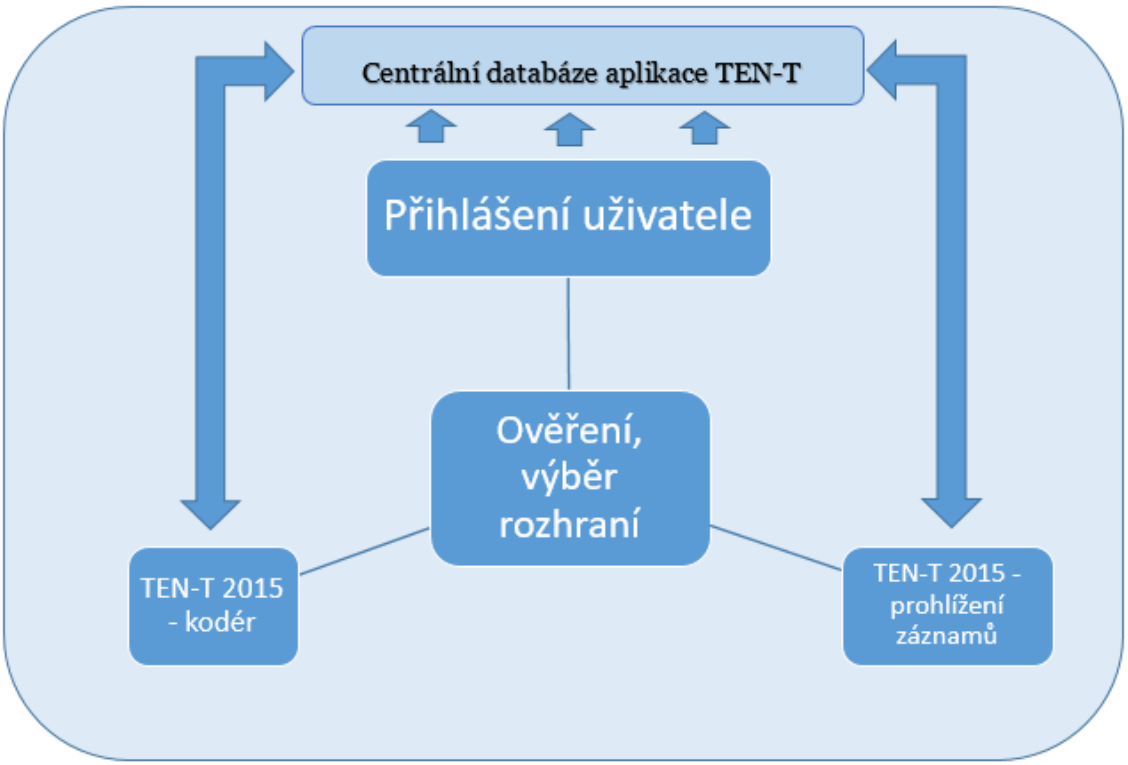
V průběhu měsíce března roku 2016 se však objevila poptávka na další audit silniční sítě a to na dalších komunikacích, případně na audity, které sledují jiné parametry či je rozšiřují (například audit zaměřený na analýzu reklamních zařízení). Z toho důvodu se několikrát měnila struktura databáze, později se dokonce jedna databáze rozpadla na několik dílčích částí až do podoby aktuální aplikace, kterou práce představím v příští podkapitole.



Obrázek 10 - Schéma původní verze databáze pro aplikaci TEN-T (Zdroj: Autor)

Jak je na schématu patrné aplikace obsahuje několik relací v databázi a to v různých kardinalitách od 1:1 (např. úsek má jen jednoho správce) až po M:N.

V závislosti na tomto schématu níže uvádím zjednodušené schéma samotné aplikace.



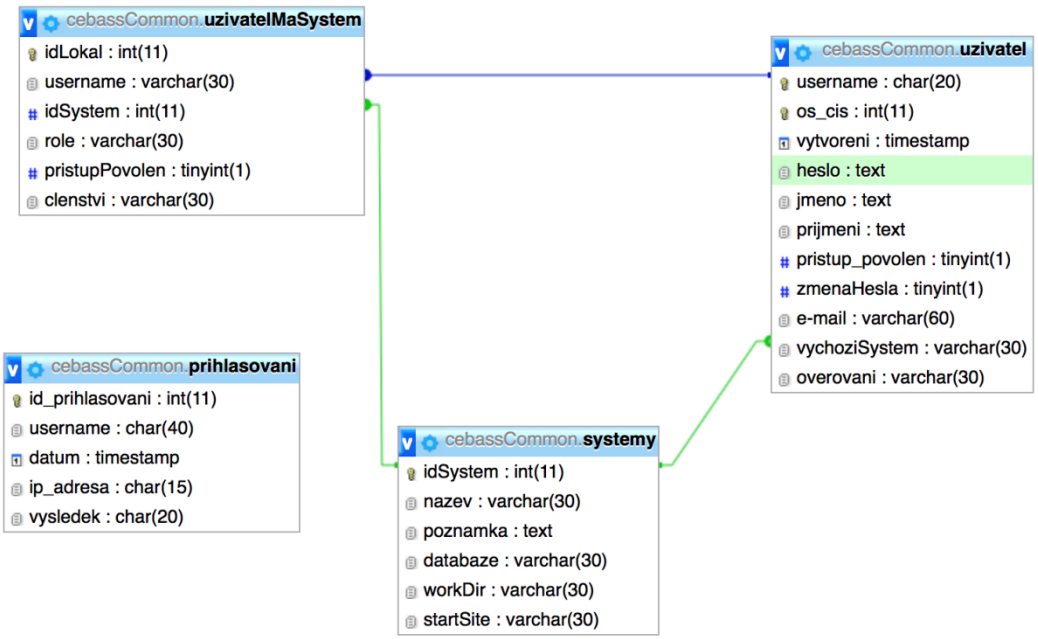
Obrázek 11 - Zjednodušené schéma aplikace (Zdroj: Autor)

Současná architektura aplikace CEBASS

Současná architektura je značně předělaná a dá se tak říci, že systém je teď modulární. To v praxi znamená, že nad jednotlivými podsystémy dřímá společná část, starající se o správu relace uživatele, správné připojení databáze, také zde nalezneme rozhraní pro změnu hesla, případně knihovny pro generování PDF a XLS (Excel) souborů.

V souvislosti s těmito změnami tak bylo původní schéma „roztrženo“ a nahrazeno několika dílčími databázemi a novými „spojovými“ tabulkami, které se starají o relace a vazbu. Analogicky stejným způsobem také byla předělána adresářová struktura.

Dále musel být předělán mechanismus ověřování uživatele, který nyní musí rozhodnout, kterou databázi připojit a se kterou uživatel skutečně může pracovat.



Obrázek 12 - Schéma společné databáze systému CEBASS (Zdroj: Autor)

Výše lze vidět první databázi, se kterou uživatel komunikuje, tato databáze z původní verze přebírá tabulku uživatelů a log přihlašování, dále zde nalezneme tabulku se seznamem podsystému a tabulku spojující systém s uživatelem.

Základní údaje o uživateli jsou pouze na jednom místě a to pro všechny systémy, v případě situace, kdy uživatel má přiřazeno více systémů se mu objeví rozhraní, kde provede výběr požadovaného systému. Pro každý systém uživatel může mít jinou roli, případně členství ve skupinách.

V tabulce systému pak nalezneme potřebné informace k systémům, název databáze příslušící danému systému, pracovní adresář (workDir), kde se nachází soubory daného systému a hodnotu startSite, což je stránka, na kterou má systém uživatele přesměrovat jako prvního.

Dále zde nalezneme databáze cebassRsd a cebassReklamy, druhé zmiňované v této práci podrobněji rozebírat nebudu.

Databáze cebassRsd pak kopíruje původní schéma databáze (Obrázek 10) pochopitelně bez části, která obsahuje údaje o uživatelích, v souvislosti s touto změnou bylo třeba také předělat SQL dotazy v databázi, které si pomocí JOIN připojují jiné tabulky, v tomto směru ale databáze MariaDB (MySQL) umožnuje

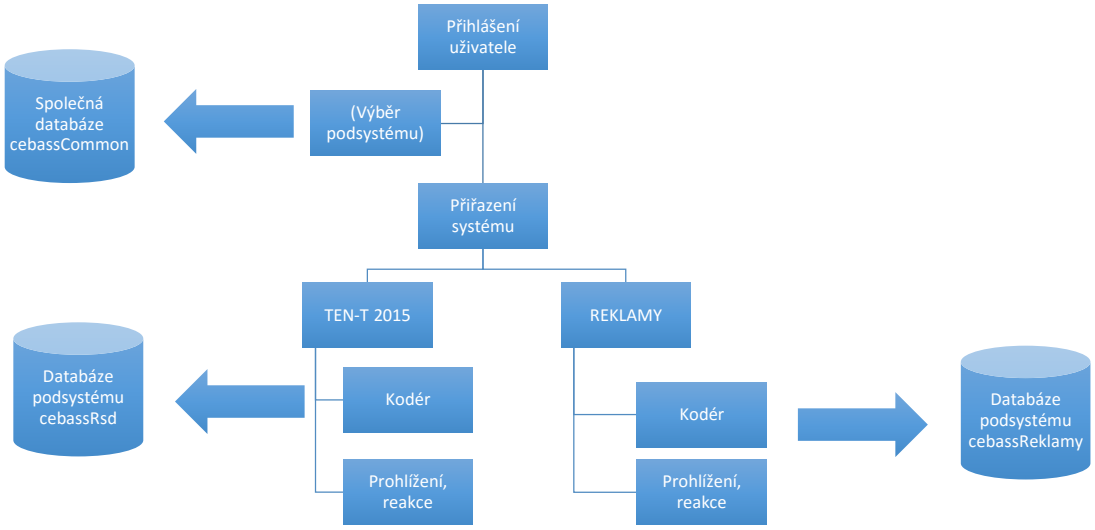
JOIN tabulek napříč databázemi a to jednoduše, před název tabulky se doplní název databáze.

Původní podoba SQL dotazu:

```
SELECT id_zaznamu, username, jmeno, prijmeni FROM zaznamy
        JOIN uzivatel USING (username)
```

Upravená podoba SQL dotazu:

```
SELECT id_zaznamu, username, jmeno, prijmeni FROM zaznamy
      JOIN cebassCommon.uzivatel USING (username)
```



Obrázek 13 – Schéma současné aplikace Cebass (Zdroj: Autor)

5. Testování aplikace

Vyčerpávající úvod do testování bezpečnosti, představení aplikace a historie aplikace již je za námi, v následující hlavní kapitole tak práce přikročí k samotnému testování na možné hrozby a následně provede implementaci přímo na aplikaci. Aplikace se nachází na webovém serveru v síti ČVUT a je dostupná pod názvem http://tent.fd.cvut.cz, případně na http://cebass.cz

Na úvod bych jen rád zmínil fakt, že aplikace vznikala poměrně progresivním tempem a vzhledem k tomu, že se jedná o výtvor pouze dvou programátorů a to navíc bez větších zkušeností na poli programování webových aplikací, je více než pravděpodobné, že aplikace určité chyby obsahuje, ale úkolem této práce není nalézt a opravit všechna její slabá místa, rozsah takové práce by byl bezpochyby daleko obsáhlejší. Představím tak pouze hlavní bezpečnostní mezery.

Zaměřím se na hlavní hrozby dle žebříčku OWASP TOP 10, postup při odhalování hrozeb budu používat takový, který je uveden v dokumentu se žebříčkem OWASP TOP 10 (30), také využiji příručky OWASP Testing Guide v4. (31)

Postup testování

V rámci testování bude práce postupovat napříč aplikací a testovat různá rozhraní na chyby, respektive hrozby, které by mohla obsahovat, tak aby práce co nejvíce zohlednila jednotlivé kategorie OWASP TOP 10. V rámci testování využiji své pozice tvůrce aplikace a testování tak bude metodou white box, tedy využiji veškeré znalosti o aplikaci v rámci testování. Pro testování jsem si vytvořil uživatelský účet „tester“, který se ověřuje vůči databázi, dále pak budu používat svůj ČVUT login, který se ověřuje vůči LDAP.

Předmětem testování tak především bude:

- Stránka s přihlašováním (A1 – Injektování, A6 – expozice citlivých dat)
- Správa relace (sezení uživatele), rozhraní pro výběr subsystému (A2 – chybná autentizace a správa relace, A4 – Nezabezpečené přímé odkazy)
- Rozhraní vkládající a zobrazující data (A3 – Cross-site Scripting XSS)
- Zabezpečení fotografií (A4 – Nezabezpečené přímé odkazy)

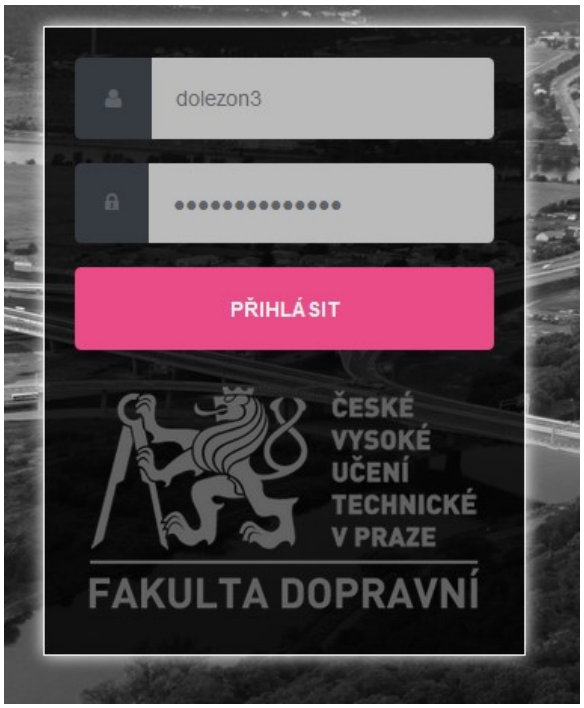
Použité nástroje

- OS Windows 10 64bit, Enterprise
- Webový prohlížeč Mozilla Firefox 45.0.2 (32)
- Java verze 8 verze 91 (33)
- OWASP ZAP (34)
- WireShark 2.0.3 (35)
- Firefox LiveHTTPHeaders (36)
- Firefox Firebug (37)
- Firefox Tamper Data (38)
- PHPStorm (39)

Testování - Stránka s přihlašováním

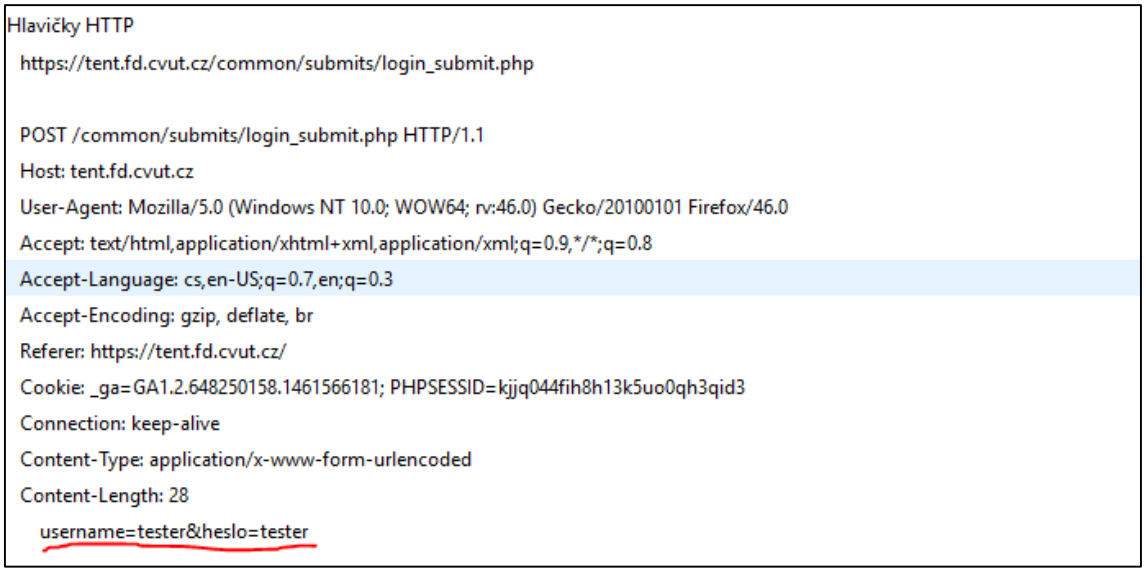
SQL Injection

Aplikace data poskytuje uživatelům pouze po přihlášení, tedy na úvodní stránce je tak přihlašovací dialog, kde se uživatel přihlásí, následně se údaje ověří vůči lokální databázi, případně LDAP adresáři.



Obrázek 14 - Přihlašovací dialog (Zdroj: Autor)

Pole pro zadání uživatelského jména a hesla nejsou dynamicky nijak kontrolována na vstupy uživatele, nicméně tato kontrola se provádí na straně serveru při zpracování přihlášení, kde probíhá kontrola délky a kontrola, zda vstup obsahuje pouze alfanumerické znaky. Formulář se zpracovává metodou POST, tedy data se odesílají uvnitř http žádosti.



Obrázek 15 - Hlavička http požadavku s daty typu POST (Zdroj: Autor)

Je vidět, že aplikace volá ke zpracování údajů soubor login_submit.php, který pak následně provede přihlášení.

```
$dbh = new PDO("mysql:host=$server;dbname=$database", $user_db, $pass_db, $options);
$dbh->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
$stmt = $dbh->prepare("
SELECT username, os_cis, jmeno, prijmeni, heslo, pristup_povolen,zmenaHesla, overovani
FROM cebassCommon.uzivatel
          WHERE username = :username
          ");
$stmt->bindParam(':username', $username, PDO::PARAM_STR);
// $stmt->bindParam(':heslo', $heslo, PDO::PARAM_STR, 40);  ldapujeme
$stmt->execute();
$vys = $stmt->fetchAll();
```

Obrázek 16 - PHP kód s SQL dotazem na výběr uživatele z DB (Zdroj: Autor)

Pokusím se za pomoci těchto znalostí provést SQL inject a získat tak určitá data z databáze. Za tímto účelem tedy změním proměnou username na „testik OR 1“. Dle mé představy by pak SQL dotaz po „dosazení“ vypadal následovně:

```
SELECT username, os_cis, jmeno, prijmeni, heslo,
pristup_povolen,zmenaHesla, overovani
```

```
FROM cebassCommon.uzivatel
WHERE username = „tester“ OR 1
```

Proti tomuto útoku je však aplikace chráněna, po pokusu se přihlásit dostanu zprávu, že se přihlášení nezdařilo, ať si zkontroluji přihlašovací údaje, nejsem schopen ani zjistit, zda se vůbec provedl dotaz vůči databázi, po ohledání zdrojového kódu jsem zjistil, že se dotaz neprovede, jelikož se zastaví už u kontroly na alfanumerické znaky (pouze tečka je v username dovolena)

```
elseif (!ctype_alnum(str_replace('.', '', $_POST['username']))) {
  $chyba = "Chyba ověřování, zkontroluj username a heslo ";
  $site = '../index.php';
  $vysledek = 'chybny vstup';
  $time = 3;
```

Obrázek 17 - Podmínka ověřující vstupy (Zdroj: Autor)

Půjdu však dál a tyto řádky budu komentovat čistě z důvodů abych zjistil, co se stane, pokud moje pozměněná proměnná username postoupí až do dotazu. Také si pomocí funkce print_r() [12]nechám vrátit dotaz, který byl použit pro výběr uživatele.

Výsledkem je opět neúspěch, jelikož díky použití ovladače PDO[13] aplikace nedovolí vůbec přiřadit za username mou konstrukci. Dotaz je parametrizovaný. V tomto směru tedy aplikace prošla na výbornou, obzvláště pokud přihlédneme k faktu, že za normálních okolností by tento pokus o inject ztroskotal v rámci ověřování na alfanumerické znaky, tedy i bez použití PDO by se útok nezdařil.

LDAP injection

Aplikace provádí ověřování vůči LDAP server v případě, že je v lokální databázi u uživatele v atributu ověřování nastaveno na “ldap”, tedy nejdříve dojde k dotazu vůči databázi, zda se vůbec zadané username nachází v databázi, pokud ano

[12] PHP funkce print_r() – provede výpis proměnné, včetně datového typu pole (40)

[13] Ovladač PDO – rozhraní pro komunikaci s SQL databází, nahrazující funkci mysql_query a jejího nástupce mysqli_query, poskytuje ochranu proti SQL injection díky možnosti „předpřipravit“ dotaz (41)

kontroluje se právě zmíněný atribut ověřování, který může nabývat hodnot „lokal“ nebo výše zmíněnou „ldap“.

Po této fázi dojde k postoupení požadavku na ověření vůči LDAPu a to velmi jednoduchým způsobem, username, které je ověřeno, díky předešlému postupu, že neobsahuje žádné nevyžádané konstrukce se připojí do řetězce a pomocí funkce ldap_bind [14]se totéž provede s heslem. Na základě návratové hodnoty (TRUE, FALSE) dojde k přihlášení uživatele nebo k odmítnutí autorizace.

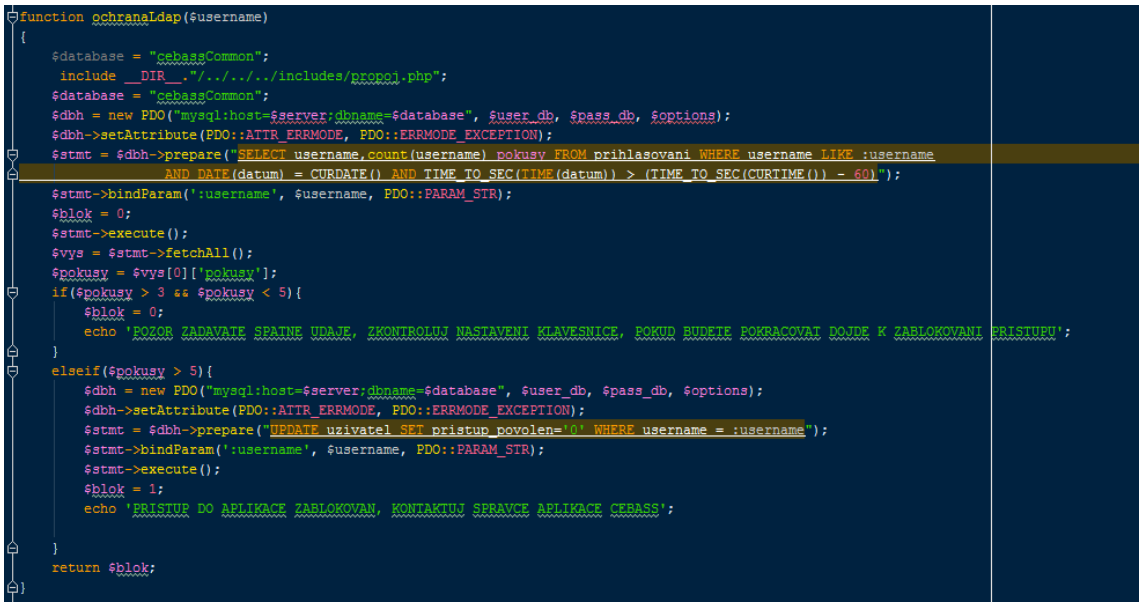
Ani zde není možnost podstrčit systému upravený dotaz, nicméně při testování jsem narazil na možnou hrozbu, která aplikaci hrozí a obzvláště u LDAP ověřování by mohly být následky poměrně velké.

Hrozba se týká útoku silou na hádání hesla uživatele (username ČVUT je veřejné přístupné, tady není co hádat). Aplikace zkrátka vyhoví každé žádosti o ověřování a mohla by tak být z tohoto pohledu zneužita.

Implementace ochrany LDAP před útokem hrubou silou

V závislosti na toto zjištění jsem v rámci této práce doprogramoval mezifázi, která se provede po ověření uživatele vůči lokální databázi, ale než se přistoupí k dotazu vůči LDAP, proběhne další dotaz vůči lokální databázi, jehož předmětem je prohledání logu přihlašování, v případě že dojde během 60 sekund k 5 přihlášením, aplikace uživatele místně zablokuje, respektive po 4. pokusu jej nejdříve varuje.

[14] Ldap_bind() – integrovaná funkce PHP, která provede připojení do LDAP adresáře na základě uživatelského jména a hesla, vrací tak hodnotu TRUE nebo FALSE (42)



```
function ochranaLdap($username)
{
    $database = "cebassCommon";
     include __DIR__."/../../../includes/propoj.php";
    $database = "cebassCommon";
    $dbh = new PDO("mysql:host=$server;dbname=$database", $user_db, $pass_db, $options);
    $dbh->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    $stmt = $dbh->prepare("SELECT username,count(username) pokusy FROM prihlasovani WHERE username LIKE :username
                AND DATE(datum) = CURDATE() AND TIME_TO_SEC(TIME(datum)) > (TIME_TO_SEC(CURTIME()) - 60)");
    $stmt->bindParam(':username', $username, PDO::PARAM_STR);
    $blok = 0;
    $stmt->execute();
    $vys = $stmt->fetchAll();
    $pokusy = $vys[0]['pokusy'];
    if($pokusy > 3 && $pokusy < 5){
        $blok = 0;
        echo 'POZOR ZADAVATE SPATNE UDAJE, ZKONTROLUJ NASTAVENI KLAVESNICE, POKUD BUDETE POKRACOVAT DOJDE K ZABLOKOVANI PRISTUPU';
    }
    elseif($pokusy > 5){
        $dbh = new PDO("mysql:host=$server;dbname=$database", $user_db, $pass_db, $options);
        $dbh->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
        $stmt = $dbh->prepare("UPDATE uzivatel SET pristup_povolen='0' WHERE username = :username");
        $stmt->bindParam(':username', $username, PDO::PARAM_STR);
        $stmt->execute();
        $blok = 1;
        echo 'PRISTUP DO APLIKACE ZABLOKOVAN, KONTAKTUJ SPRAVCE APLIKACE CEBASS';

    }
    return $blok;
}
```

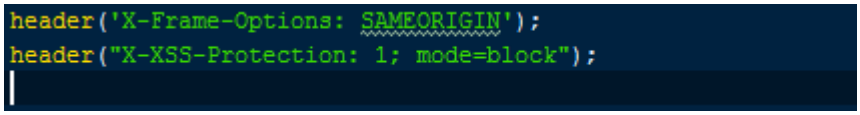
Obrázek 18 - Implementace ochrany LDAP před útokem hrubou silou (Zdroj: autor)

Využití nástrojů OWASP

Na stránku s přihlašováním, v tomto případě na stránku index.php., jsem využil také nástroje pro testování bezpečnosti, které poskytuje OWASP sám, například OWASP ZAP.

Po ukončení testování jsem obdržel varování před nedefinovanou ochranou před XSS útoky, které jsou na seznamu OWASP TOP 10, nicméně toto varování je ohodnoceno rizikem „nízké“.

Rizikem středním však je ohodnocena chyba, týkající se nedefinované politiky X-Frame-options (43), která umožnuje zneužití za pomocí sezení uživatele a kliknutí na jinou stránku. Obě varování však byla odstraněna úpravou definice hlavičky, a to za pomocí PHP, následovně:



```
header('X-Frame-Options: SAMEORIGIN');
header("X-XSS-Protection: 1; mode=block");
```

Obrázek 19 - Nastavení hlaviček pomocí PHP

Správa relace (sezení uživatele), rozhraní pro výběr subsystému

Další věcí, která bude podrobena testování a rozboru je správa relace uživatele, navíc v rámci této části provedu testování i na rozhraní, kde uživatel může vybrat subsystém, právě při přihlášení a výběru systému se totiž inicializují proměnná sezení, která jsou pak navíc používány napříč aplikací.

Při přihlášení uživatele jsou do relace uživatele načteny informace, které jsou potřebné pro pozdější provoz aplikace, tyto informace jsou navíc v průběhu pohybu uživatele porovnávány, zda je jejich obsah konzistentní. Z bezpečnostních důvodů je také do sezení ukládán, v zašifrované formě, otisk prohlížeče a také IP adresa uživatele, tyto informace pak jsou porovnávány a v případě neshody je sezení ukončeno. Právě tyto opatření mají za úkol ztížit ukradnutí sezení a dostat se tak datům.

```
$_SESSION['prijmeni'] = $user_prijmeni;
$_SESSION['jmeno'] = $user_jmeno;
$_SESSION['username'] = $username;
$_SESSION['ip_sezeni'] = sha1($_SERVER['REMOTE_ADDR']);
$_SESSION['agent'] = sha1($_SERVER['HTTP_USER_AGENT']);
$_SESSION['user_id'] = $user_id;
```

Obrázek 20 - Ukázka inicializace některých proměnných sezení (Zdroj: Autor)

Během testování byly objeveny následující zranitelnosti systému správy relace:

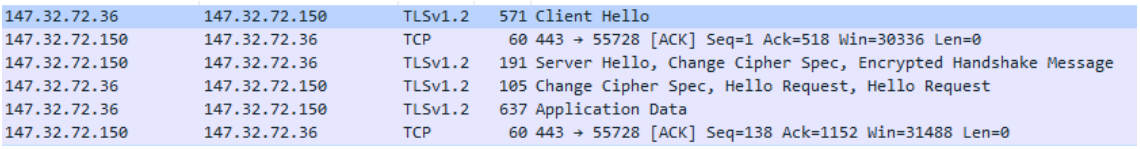
1. Není řádně nastavena délka sezení, respektive její kontrola ani platnost souborů cookies[15]
2. Odesílání souborů cookies není vynuceno šifrovaným spojením
3. Existují sice funkce ověřující otisk prohlížeče a IP adresu, ale informace nejsou při přihlášení ukládány například do databáze, tedy pří ukradnutí sezení si útočník může otisk prohlížeče změnit a kontrola IP nemusí být

[15] Cookies – data uložená u klienta obsahující některá uživatelská data o sezení

spolehlivá, například kvůli funkci NAT[16], navíc tyto informace jsou hašovány dnes již relativně slabou funkcí SHA1 (44)

4. Aplikace neověřuje práva uživatele k přístupům k jednotlivým subsystémům

Aplikace naopak v jiných věcech uspěla, například odesílání přihlašovacích údajů a komunikace jako taková je vynucena přes rozhraní HTTPS, tedy je šifrována, stejně jako komunikace s LDAP server probíhá přes šifrovaný protokol (LDAPS), tento stav jsem dokázal poslouchánim mého síťového rozhraní za pomocí programu WireShark.



Obrázek 21 - Ukázka šifrovaného spojení vůči webovému serveru (Zdroj: Autor)

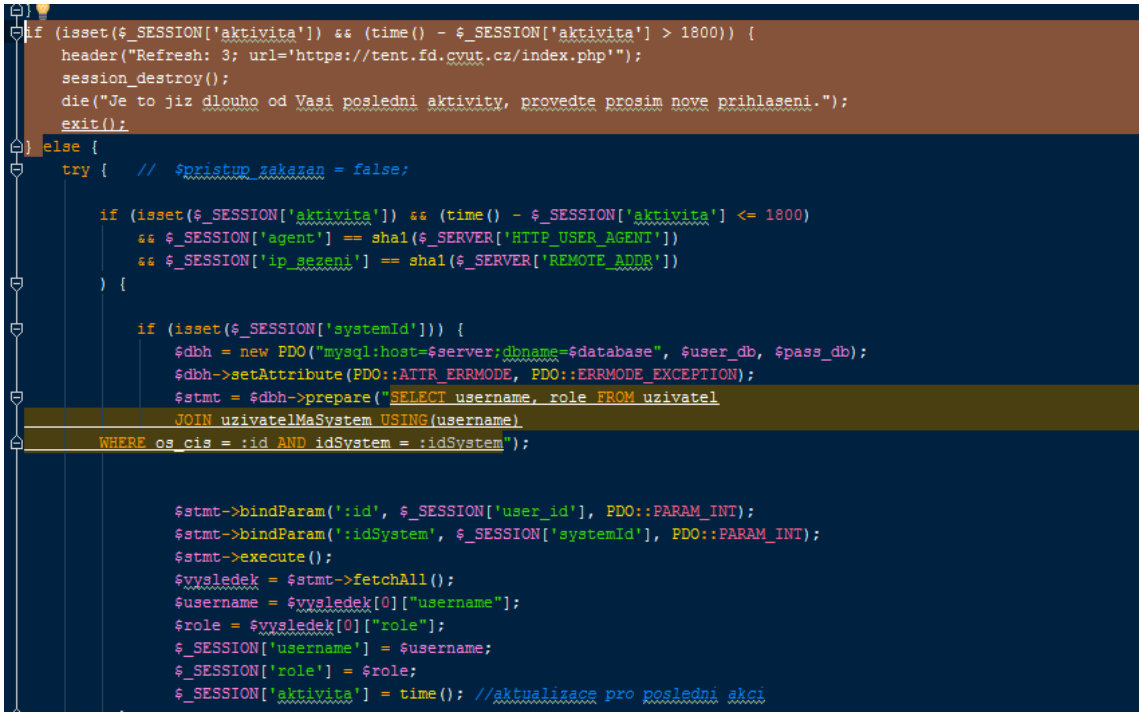
Implementace oprav

V následujících řádcích provedu opravy jednotlivých výše uvedených problémů a to postupně, jak jsou uvedeny.

1. Problém se dá řešit definováním platnosti souborů cookies a to přímo v konfiguračním souboru php.ini, konkrétně se jedná o volbu ***session.cookie_lifetime***. Ve výchozím stavu je tato doba nastavena na 0, což znamená neomezenou dobu, tedy do zavření prohlížeče, nutno však dodat, že tento parametr nemá na sezení, respektive na proměnné sezení vytvářené za pomocí PHP, přímý vliv. I v případě propadnutí platnosti souborů cookie bude sezení PHP nedotčeno. Aplikoval jsem tak následující konstrukci, která se postará o to, aby aplikace na straně klienta sama ukončila sezení a odstranila data z proměnných.

[16] NAT – překlad adres, jehož výsledkem v našem pojetí může být fakt, že dva různé počítače mají stejnou adresu IP, jelikož jsou například na vnitřní síti a brána na rozhraní této sítě provádí NAT, budeme tak mít informaci pouze o IP adrese této brány

Vytvořím si jednoduše další proměnnou, která se bude načítat aktuálním časem. V rozhraní, které kontinuálně při uživatelově aktivitě provádí kontrolu jeho sezení, budu tuto hodnotu kontrolovat, respektive pokud se splní podmínka, že je aktuální čas, kdy probíhá kontrola sezení a od toho odečtená hodnota této proměnné větší než 30 minut, provede se ukončení sezení, v opačném případě se hodnota poslední aktivity zaktualizuje aktuálním časem.



Obrázek 22 -Implementace opravy, ukončení sezení po určitém čase (Zdroj: Autor)

2. Zde je oprava naprosto triviální, opět provedu úpravu konfiguračního souboru PHP. Konkrétně měním parametr ***session.cookie_secure*** (45), který je ve výchozím stavu nastaven na hodnotu false (nepravda).
3. Zde je hrozba poměrně značná a také implementace opravy bude náročnější, můj postup bude následující:
    - Využiji při přihlášení log přihlašování, který využívá SQL databázi, do této DB otisku prohlížeče, IP adresu už log eviduje, byť ne v zašifrované podobě
    - Při každém ověřování uživatele, které se provádí na každé stránce, budou tyto hodnoty ověřeny, po ověření se provede regenerování session_id

- Hodnoty budou využívat šifrovací funkci SHA256 (46) ale i původní funkci SHA1
- Ověření bude na více úrovních, ověřovat se bude jak aktuální stav se stavem v DB, tak i s hodnotou uloženou na serveru v sezení
- V případě zjištěného problému se tato událost zaeviduje do databáze s konkrétním důvodem

```php
function overovaniRelace(){
    include (__DIR__."/zapis_log.php");
    include (__DIR__."/propoj.php");

    if($_SESSION['agent'] == sha1($_SERVER['HTTP_USER_AGENT'] ) && $_SESSION['ip_sezeni'] == sha1($_SERVER['REMOTE_ADDR'] )){
            $database = 'cebassCommon';
            $dbh = new PDO("mysql:host=$server;dbname=$database", $user_db, $pass_db);
    $dbh->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    $stmt = $dbh->prepare("SELECT username, ip_adresa, agent FROM prihlasovani WHERE username = :username
                        AND id_prihlasovani = (SELECT MAX(id_prihlasovani) FROM prihlasovani WHERE username = :username )");
    $stmt->bindParam(':username', $_SESSION['username'], PDO::PARAM_STR);
    $stmt->execute();
    $vysledek = $stmt->fetchAll();
        if(hash('sha256',$_SERVER['HTTP_USER_AGENT']) == $vysledek[0]['agent'] && $_SERVER['REMOTE_ADDR'] == $vysledek[0]['ip_adresa']) {
            session_regenerate_id();
            session_regenerate_id(true);
            $_SESSION['aktivita'] = time(); //aktualizace pro posledni akci
            return TRUE;
        }
        else{
            $vysledek = "Informace uživatele se neshodují s daty v databázi! Ukončuji relaci!";
            zapis_log_login($_SESSION['username'], $vysledek, $_SERVER['REMOTE_ADDR'], $_SERVER['HTTP_USER_AGENT']);
            return FALSE;
        }
}
    else{
        $vysledek = "Informace uživatele se neshodují s daty v relaci uživatele! Ukončuji relaci!";
        zapis_log_login($_SESSION['username'], $vysledek, $_SERVER['REMOTE_ADDR'], $_SERVER['HTTP_USER_AGENT']);
        return FALSE;
    }
}
```

Obrázek 23 - Implementace pokročilé kontroly relace uživatele (Zdroj: Autor)

Díky výše uvedeným změnám bude pro potencionálního útočníka nemožné „ukrást“ uživateli jeho relaci, značně ztížený také bude mít přístup k datům relace jako takovým.

4. Tuto chybu nejdříve lehce rozvedu, chyba totiž koresponduje s chybou týkající se nezabezpečeného objektu, který je uveden v OWASP TOP 10, jednoduše pokud uživatel ručně změní pracovní adresář v adresním řádku, aplikace mu tuto obrazovku v zásadě ukáže, byť funkčnost tohoto podsystému bude omezena, jelikož informace v relaci uživatele odpovídají jinému podsystému (například název databáze).

idSystem	nazev	poznamka	databaze	workDir	startSite
2	TEN-T 2015	Analýza silniční sítě zjišťující deficity na komun...	cebassRsd	rsd	rozcestnik.php
3	Reklamy	Systém pro analýzu reklamích zařízeních na vybraný...	cebassReklamy	reklamy	
4	TEN-T KODÉR	Rozhraní pro zanášení deficitů TENT	cebassRsd	vip	vip.php
5	Reklamy Kodér	Rozhraní pro zanášení deficitů týkající se reklam	cebassReklamy	reklamy	form.php

Obrázek 24 - Náhled na tabulku se systémy (Zdroj: Autor)

Nabízí se však poměrně jednoduché řešení, a to kontrola adresního řádku a to taková, která bude kontrolovat, zda se uživatel pohybuje v pracovním adresáři subsystému, který si skutečně vybral, případně který má jako jediný určený. Tuto část dám opět jako součást ověřování a bude tak na každé stránce. Výjímka se bude pochopitelně stahovat na složky common a pdf, kam může každý.

```php
function overSystem()
{
    if(isset($_SESSION['systemId'])) {
        require_once(__DIR__ . "/../html/common/includes/funkce.php");
        $systemID = $_SESSION['systemId'];
        $systemInfo = najdiSystem($systemID);
        $uri = explode('/', $_SERVER['REQUEST_URI']);
        $prvniAdresar = $uri[1];
        if ($prvniAdresar == $systemInfo[0]['workDir'] OR $prvniAdresar == 'common' OR $prvniAdresar == 'pdf') {
            return TRUE;
        } else {
            require_once(__DIR__ . "/zapis_log.php");
            $vysledek = "Uživatel se nachází ve špatném adresáři! Ukončuji relaci!";
            zapis_log_login($_SESSION['username'], $vysledek, $_SERVER['REMOTE_ADDR'], $_SERVER['HTTP_USER_AGENT']);
            return FALSE;
        }
    }
    return FALSE;
}
```

Obrázek 25 - Ukázka implementace kontroly pracovního adresáře (Zdroj: Autor)

Rozhraní vkládající a zobrazující data

Předmětem testování jsou rozhraní, která umožnují uživateli vkládat nějaká data, v našem případě se jedná o možnost vkládat k jednotlivým záznamům tzv. reakci, tuto reakci si pak může každý uživatel zobrazit, případně její autor i editovat a smazat.

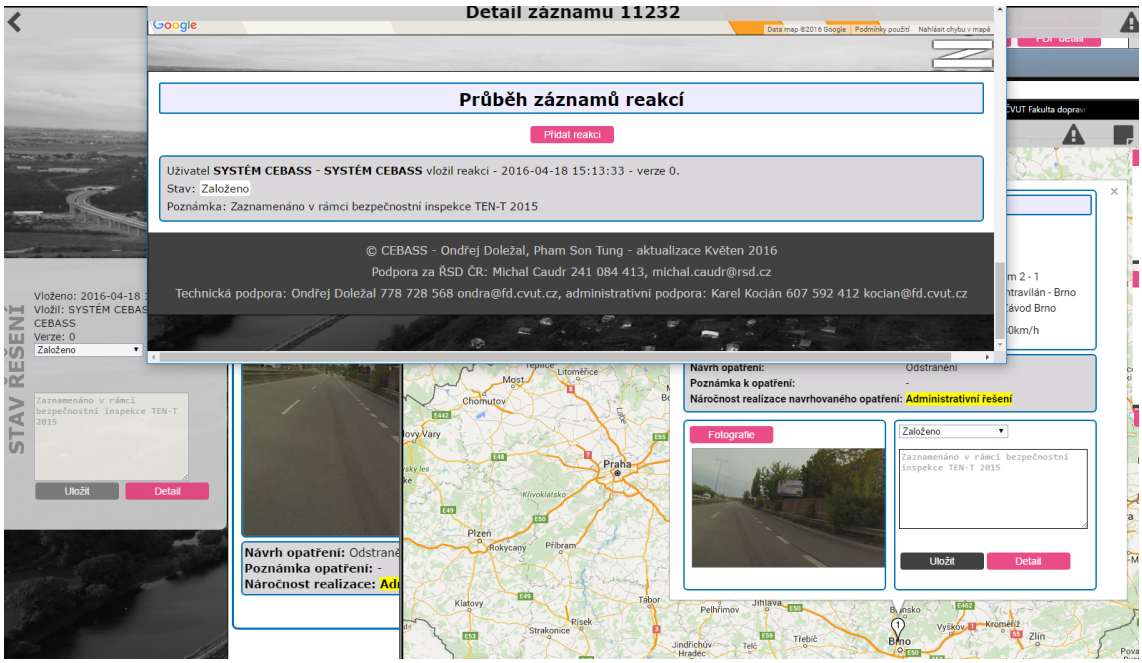
Právě toto rozhraní teoreticky dovoluje existenci problému, vycházejícího z nedostatečné kontroly vstupů uživatele a dalo by se tak lehce zneužít například pro Cross-site Scripting (XSS) (47), který je zmíněn v OWASP TOP 10.

Uživatel má v této reakci možnost vyplnit poznámku a pak z rozevíracího seznamu vybrat i fázi řešení, v případě stavu, kdy uživatel zadá „Odstraněno“ mu navíc přibude možnost nahrát fotografii.

Aplikace nabízí několik možnosti vkládat reakce na deficity, celkový počet rozhraní, které jsme schopni v aplikaci nalézt je 3.

Vkládat je možné z:

- Rozhraní klasického výpisu
- Rozhraní zobrazující záznam v interaktivní mapě
- Rozhraní zobrazující detail záznamu (v tomto rozhraní je navíc možné si zobrazit kompletní výpis vložených reakcí)



Obrázek 26 - Ukázka všech možných rozhraní pro vkládání reakcí (Zdroj: Autor)

Dobrá zpráva však je, že všechna tato rozhraní, byť každé vypadá trochu jinak, využívají pro ukládání dat, respektive komunikaci s databází, stejný modul, který je volán pomocí technologie AJAX (která také data vyplněná uživatelem posílá pomocí POST)

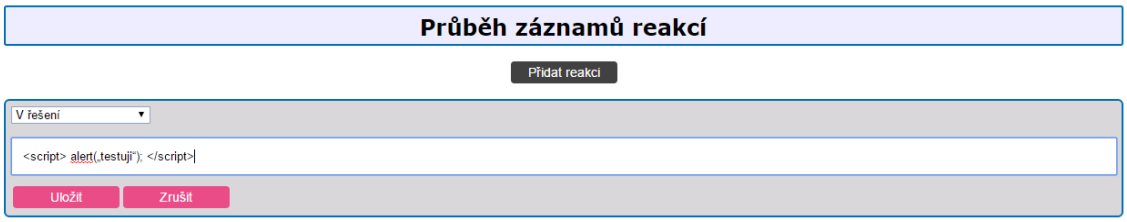
Přistupme k testování, zaměřím se právě na hrozbu týkající se XSS, která byla zmíněna výše. Pokusím se k záznamu vložit jednoduchý kód, napsaný v jazyce JavaScript, který zobrazí okno s oznámením „TESTUJI!“, jeho syntaxe bude následující:

```
<script>

alert(„testuji“);

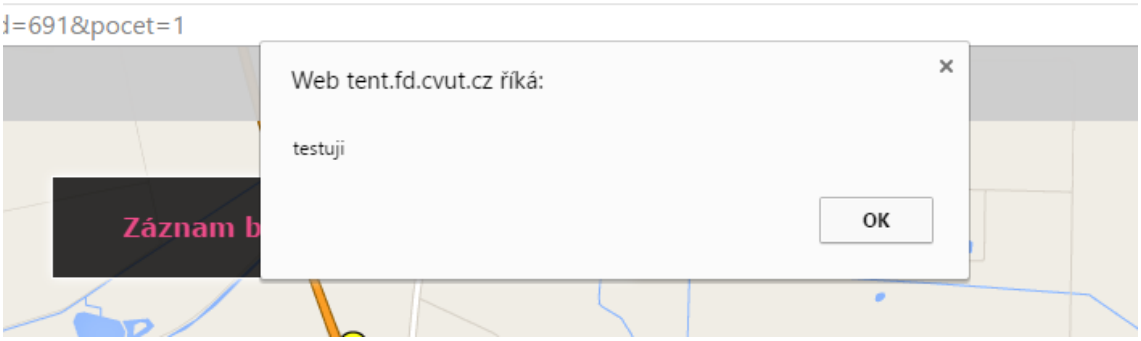
</script>
```

Pro vkládání například využiji okno s detailem záznamu.



Obrázek 27 - Vkládání reakce v rozhraní detailu (Zdroj: Autor)

Ihned po uložení, kdy se objeví okno s potvrzením, že reakce byla vložena se objeví okno s obsahem „testuji, které vyvolal JavaScript.



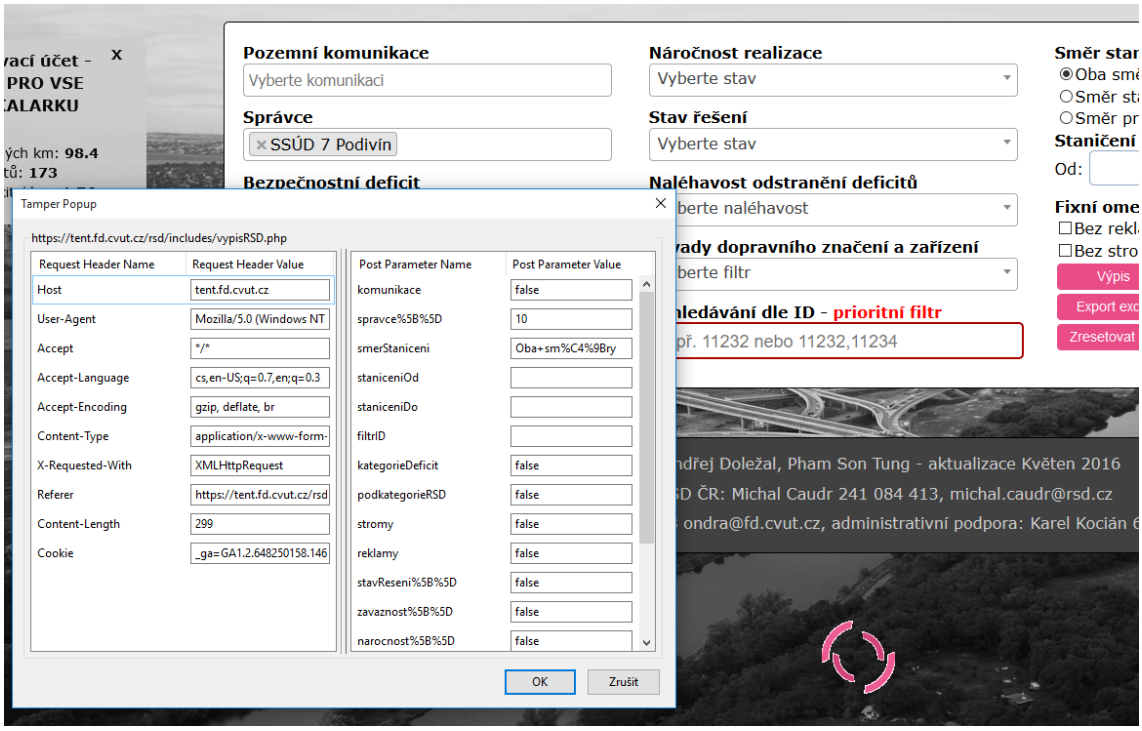
Obrázek 28 - Následek úspěšného útoku (Zdroj: autor)

Je tedy evidentní, že aplikace není chráněna proti tomuto druhu útoku. Při vstupu nedochází ke kontrole, ani se nekontrolují data, která se načítají a zobrazují z databáze. Následky této fatální bezpečnostní mezery mohou být velké, jak už bylo řečeno v úvodu této práce.

Implementace opravy

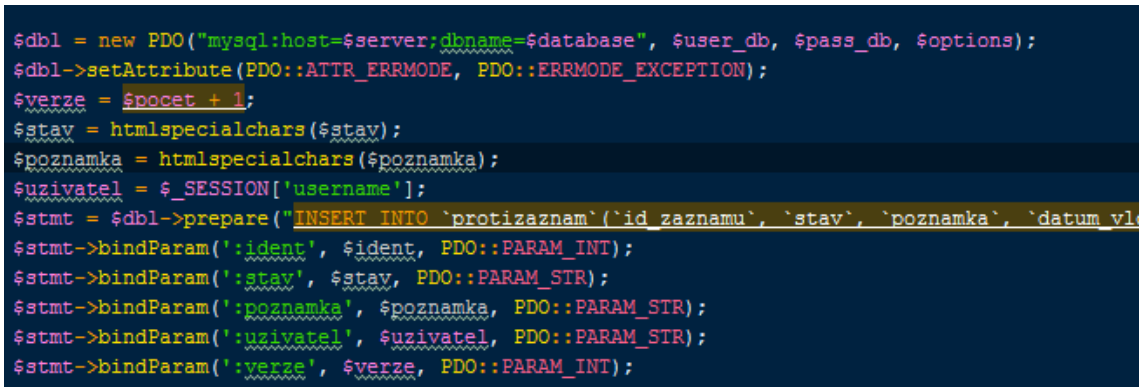
Implementace opravy je v tom případě postavena na tom, že než začnu s hodnotou, kterou uživatel zadal operovat, zkontroluji ji a pak teprve předám ke zpracování, respektive k uložení do databáze. Pro jistotu podobný přístup budu aplikovat i v opačném případě, tedy kdy si databázi říkám o hodnotu a pak ji vypisuji v prohlížeči.

Kouzlo kontroly je v tom, že použiji integrovanou funkci v PHP htmlspecialchars, která provede kontrolu textu a vrátí ji zpět „očištěnou“ od speciálních html znaků, které by v normálním případě prohlížeči říkaly, aby daný řetězec zpracoval. Funkce však neočistí text tím způsobem, že by tyto znaky odstranila, na webu se přirozené znaky jako <, >, ‘ atd. zobrazují naprosto běžně, vtip je v tom, že aby prohlížeč tyto znaky sice zobrazil, ale nepracoval s nimi, nahradí je zástupným „kódem“ (48). V tomto případě také nesmíme zapomenout na vstup „stav“, který se vybírá z roletkového menu, lze sice namítnout, že data která se přenášejí pomocí POST lze označit za každé situace za validní a tedy bezpečná, není tomu tak zcela pravda, pomocí například aplikace Tamper data (49) lze měnit hodnotu dat, posílající přes POST.



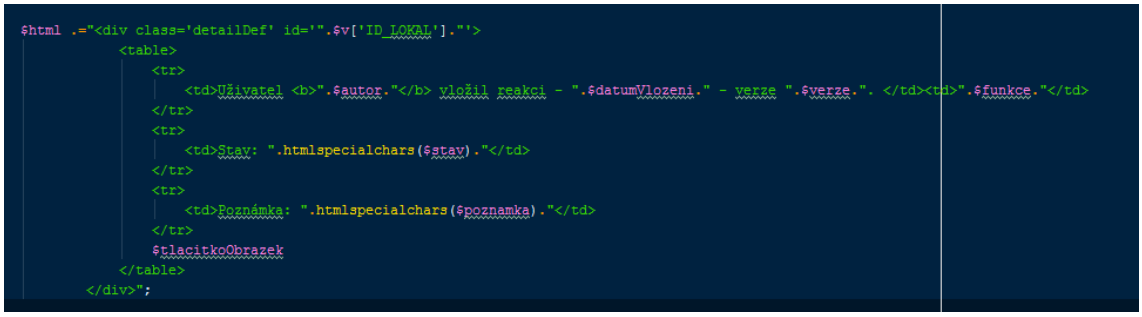
Obrázek 29 - Ukázka použití doplňku Tamper Data, který dokáže měnit obsah dat posílaných přes POST (Zdroj: Autor)

Aplikace Tamper data totiž po zapnutí sleduje aktivitu prohlížeče, respektive zachycuje http i https požadavky, díky přímému propojení s prohlížečem mu nevadí, ani právě za normálních okolností, šifrovaná komunikace přes https. V případě zaevidování požadavku, který odesílá data pomocí metody POST, požadavek „zastaví“ a nabídne uživateli možnost editovat odesílané hodnoty.



Obrázek 30 -Implementace funkce htmlspecialchars() před uložením do DB (Zdroj: Autor)

Díky společnému rozhraní, které provádí ukládání informací do databáze, nám stačí zásah jen na jednom místě. Obdobným způsobem provedeme opravu na straně, která data interpretuje, na rozdíl od předešlého případu je potřeba projít každé rozhraní, které data ukazuje.



Obrázek 31 - Oprava na straně výpisu dat z DB (Zdroj: Autor)

Zabezpečení fotografií

Dostali jsme se k poslední části, která se týká dat uložených na serveru, konkrétně fotografií, které jsou uloženy na serveru a vždy se vztahují k danému deficitu. Tyto fotografie pak může, buď ve výpisu, případně v interaktivní mapě, vidět pouze přihlášený uživatel.

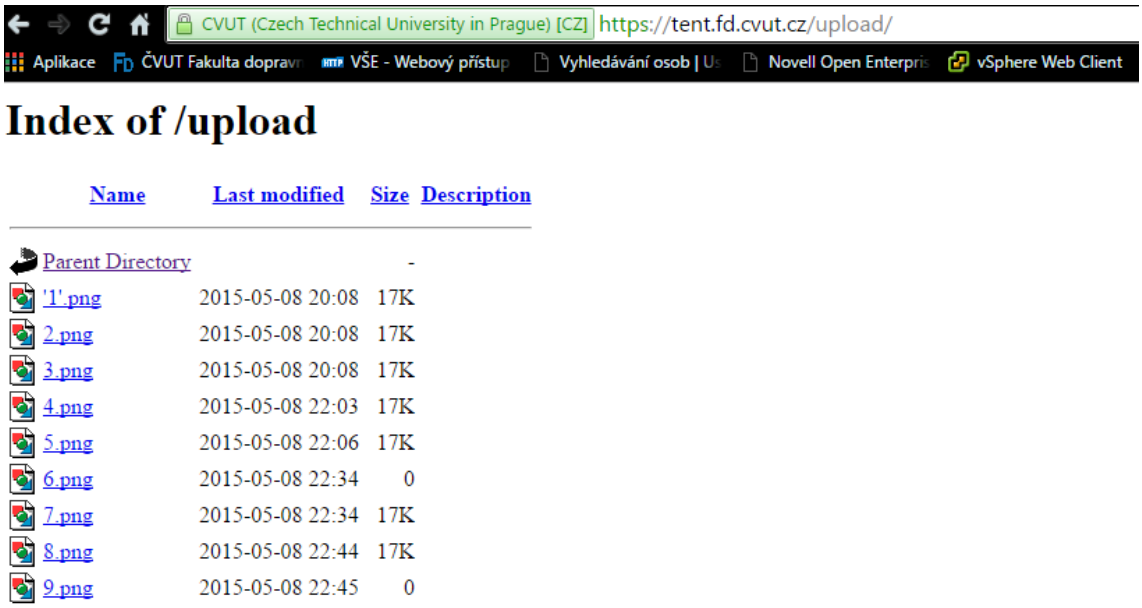
Nicméně v rámci testování byl zjištěn problém v tom, že obrázky, jako takové, jsou uloženy na webovém serveru pod jeho pracovním adresářem, tedy po zadání přímé URL adresy do vyhledávače je může vidět i nepřihlášený uživatel. Tato chyba tedy koresponduje s problematikou týkající se nedostatečně chráněných objektů. Tento stav je totiž nežádoucí, jakýkoliv člověk by si tak mohl zobrazit, respektive stáhnout, všechny fotografie za pomocí jednoduchého skriptu. Obrázky jsou totiž uložené pod číslem, které je stejné jako číslo záznamu.

Dobrá zpráva však je, že v samotném adresáři nelze listovat (pokus skončí chybou 403 – Přístup odmítnut) a zobrazit si tak seznam obrázků, to díky pravidlu, které je definováno v konfiguračním souboru pro server Apache.

```
<Directory /var/www/tent.fd.cvut.cz/html/upload>
<------>Options FollowSymLinks.
<------>AllowOverride All
</Directory>
```

Obrázek 32 - Nastavení pro složku upload (Zdroj: Autor)

Za normálních okolností je výchozí stav jiný a to takový, že v parametru Options kromě FollowSymLink se nachází ještě hodnota Indexes a právě za takových okolností by bylo možné procházet adresář.



Obrázek 33 - Příklad, kdy parametr Options obsahuje hodnotu Indexes (Zdroj: Autor)

Tato ochrana je však nedostatečná, je třeba odkazy na obrázky nějakým způsobem zabezpečit a vztáhnout na ně kontrolu sezení uživatele.

Implementace opravy

V první řadě využiji jednoduchého triku a složku s obrázky přesunu „před“ kořenový pracovní adresář webového serveru, ten v současné době má adresu /var/www/tent.fd.cvut.cz/html. Právě ve složce html se nachází složka upload, tuto složku tedy přesunu na nové umístění do /var/www/tent.fd.cvut.cz/. Po této změně již obrázky nemohou být zpřístupněny přímým odkazem, neboť nejsou v úložišti pracovního adresáře webového serveru (ten začíná o úroveň výše).

Půjdu však ještě dále a obrázky, respektive data obrázku, si nechám vrátit pomocí PHP funkce, v atributu src tak nebude odkaz na obrázek jako takový, ale jeho data (URI[17]). Použiji základ, jež je uveden na webové stránce Davida Walshe, který se zabývá vývojem webových aplikací (50).

```php
// A few settings
$image = 'cricci.jpg';

// Read image path, convert to base64 encoding
$imageData = base64_encode(file_get_contents($image));

// Format the image SRC:  data:{mime}; base64,{data};
$src = 'data: '.mime_content_type($image).'; base64,'.$imageData;

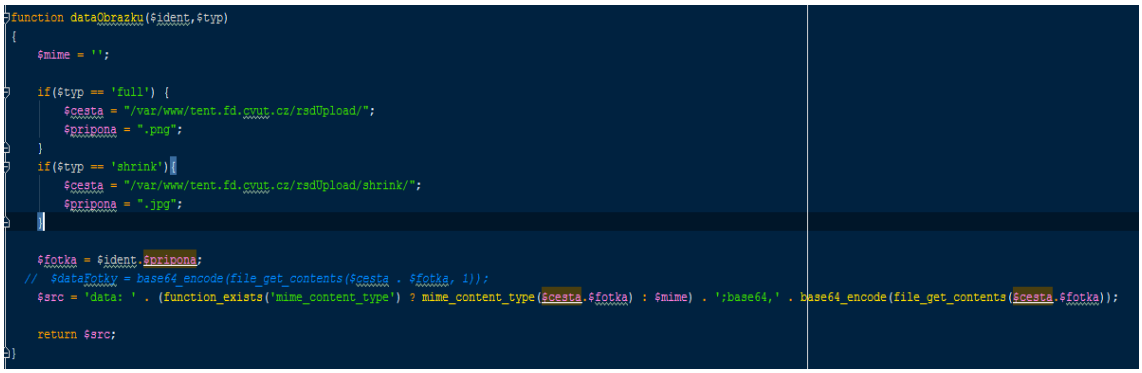
// Echo out a sample image
echo '<img src="', $src, '">';
```

Obrázek 34 - Původní funkce na výpis fotky jako dat (Zdroj: 50)

Pro mé potřeby následující řádky upravím a udělám z ní funkci, kterou budu volat vždy, když budu potřebovat data obrázku. Navíc aplikace má dva typy fotografií, jednou ve velkém rozlišení s příponou PNG, podruhé v menším rozlišení a s příponou JPG. To kvůli tomu, aby při generování PDF server zpracovával co nejméně dat, rozdíl ve velikosti totiž je propastný, zatímco obrázek v plné kvalitě má kolem 1 MB, tak zmenšený JPG má kolem 70 kB, nemusím ani zmiňovat, že při generování 1000 záznamů do PDF, je tento rozdíl markantní.

Tyto požadavky zohledním ve funkci jako další argument, navíc zmenšené obrázky se nacházejí v jiném adresáři. Má funkce, která převádí obrázek na data, bude následující:

[17] Data URI – jedná se o způsob jak zapsat obsah souboru přímo do webových stránek, případně do CSS souboru, v HTML uvozujeme tento druh vkládání souborů spojením „data:“, výhodu představuje fakt, že obrázek teoreticky nemusí být nikde nahrán a také fakt, že použití této technologie snižuje počet http žádostí při načítání stránky (51)



```php
function dataObrazku($ident,$typ)
{
    $mime = '';

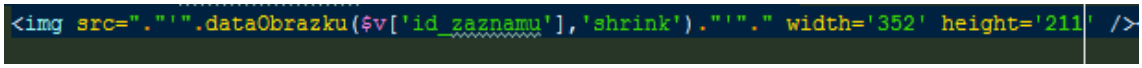
    if($typ == 'full') {
        $cesta = "/var/www/tent.fd.cvut.cz/rsdUpload/";
        $pripona = ".png";
    }
    if($typ == 'shrink'){
        $cesta = "/var/www/tent.fd.cvut.cz/rsdUpload/shrink/";
        $pripona = ".jpg";
    }

    $fotka = $ident.$pripona;
  //  $dataFotky = base64_encode(file_get_contents($cesta . $fotka, 1));
    $src = 'data: ' . (function_exists('mime_content_type') ? mime_content_type($cesta.$fotka) : $mime) . '; base64,' . base64_encode(file_get_contents($cesta.$fotka));

    return $src;
}
```

Obrázek 35 - Vlastní implementace funkce převádějící obrázek na data (Zdroj: Autor)

Funkci si jednoduše posléze zavolám z výpisu s argumentem identifikačního čísla záznamu a za typ dosadím jaký obrázek požaduji.



```php
<img src="."'".dataObrazku($v['id_zaznamu'],'shrink')."'"." width='352' height='211' />
```

Obrázek 36 - Příklad volání funkce z rozhraní exportu do PDF (Zdroj: Autor)

Nesmím ještě opomenout, že danou funkci si uložím do centrálního souboru, kde se nachází všechny ostatní funkce PHP, tento soubor se pak vkládá na začátek každého souboru, kde je třeba, ale především je chráněn funkcí, ověřující relaci uživatele, tedy je nedostupný bez přihlášení.

6. Závěr

Práce splnila všechny cíle, které byly definovány v úvodu práce. Hlavní cíl práce byl naplněn v kapitole „Testování aplikace“. V této kapitole bylo provedeno testování aplikace a to za pomocí materiálu a nástrojů, které poskytuje organizace OWASP. V rámci testování byly objeveny zranitelnosti, které byly klasifikovány jako relativně závažné, jedna z nalezených chyb dokonce dovolovala cizím uživatelům přistupovat k datům, respektive stahovat si fotografie, aniž by bylo po nich vyžadováno přihlášení. Tento problém byl vyřešen jednoduchým řešením, které spočívalo v přesunu adresáře mimo pracovní adresář webového serveru, dalším přínosem bylo místo načítání obrázku, načítání pouze jeho dat, což přineslo zlepšení na poli rychlosti načítání, mimo to i toto řešení přineslo výhodu, která spočívá v tom, že aplikace ve zdrojovém kódu neprozrazuje umístění obrázků. Velký bezpečnostním problém byl objeven u rozhraní pro vkládání

reakcí, které nekontrolovalo vstupy od uživatele, čímž otevíralo možnost pro XSS útoky, které by mohly naprosto ochromit provoz aplikace, případně by tyto útoky mohly vyústit v únik dat z aplikací. Naopak ochrana proti SQL, ale i LDAP injection je na vysoké úrovni z důvodu kontroly vstupů, které se pak používají jako argumenty klauzule WHERE, navíc se pro komunikaci s databází používá ovladač PDO, který poskytuje možnost předpřipravit si dotaz k databázi a sám o sobě zamezuje SQL injectům. Dílčí cíle práce byly také splněny, byla představena aplikace, její účel, vývoj a také její zázemí. V rámci práce jsem zmínil několik užitečných zdrojů, ze kterých jsem čerpal informace v rámci své práce a doufám, že jsem tak poskytl základní informační základnu pro čtenáře/vývojáře, pro které by tato práce mohla být inspirací při jejich vývoji.

Do budoucna si jako autor kladu za úkol aplikaci dále rozvíjet a to jak po funkční stránce jelikož aplikace s omezenou funkcionalitou nebo špatným ovládáním nebude mít úspěch u uživatelů a aplikace bez uživatelů je rovnou odkázána k neúspěchu. Pochopitelně však se bude kontinuálně pracovat na bezpečnosti aplikace, která však, jak testování ukázalo, není na tak špatné úrovni, jak jsem vzhledem k malým zkušenostem a rychlým vývojem aplikace očekával. Práce mě naučila brát bezpečnost jako součást jejího vývoje a ne jako něco, co má splňovat nad rámec. Je mnohem jednodušší v konečném případě na ni při vývoji klást důraz, než zpětně opravovat chyby. Dále mne utvrdila v tom, že obrovská část bezpečnosti je založena na tom, že řádně kontrolujeme vstupy uživatele a to za všech okolností a pokud je tato podmínka splněna, máme z velké části problém vyřešen.

7. Používané termíny

Termín	Zkratka	Význam
HyperText Markup Language	HTML	Značkovací jazyk pro tvorbu webu (Definice Autora)
Centrální evidence pro bezpečnostní analýzu silniční sítě	CEBASS	Aplikace sloužící k evidenci bezpečnostních deficitů na silniční síti (Definice Autora)
PHP: Hypertext Preprocessor	PHP	Skriptovací jazyk využívaný při vývoji webových aplikací, zpracován serverem (Definice Autora)
Ředitelství silnic a dálnic ČR	ŘSD ČR	Státní organizace zřízená Ministerstvem dopravy, spravující silniční sítě v ČR
Trans European Transport Networks	TEN-T	Evropská síť komunikací
Structured Query Language	SQL	Strukturovaný jazyk využívaný pro databáze (Definice Autora)
Lightweight Directory Access Protocol	LDAP	Protokol pro ukládání a čtení informací, data jsou uložena ve stromové struktuře (Definice Autora)
Open Web Application Security Project	OWASP	Organizace založena za účelem zlepšit míru zabezpečení webových aplikací (Definice Autora)
JavaScript	JS	Skriptovací jazyk využívaný při vývoji webových aplikací, zpracován klientem (Definice Autora)

PHP Data Objects	PDO	Rozhraní pro komunikaci s databází, nástupce mysqli_query (Definice Autora)
Secure Sockets Layer	SSL	Protokol umožňující zabezpečenou komunikaci pomocí šifrování (Definice Autora)

8. Seznam použitých zdrojů

1 VYHLÁŠKA ze dne 20. října 2011, kterou se mění vyhláška č. 104/1997 Sb., kterou se provádí zákon o pozemních komunikacích, ve znění pozdějších předpisů. In: Sbírka zákonů České republiky. 2011, ročník 2011, 317 / 2011. Dostupné z: http://www.zakonyprolidi.cz/cs/2011-317

2 *SMĚRNICE EVROPSKÉHO PARLAMENTU A RADY 2008/96/ES ze dne 19. listopadu 2008 o řízení bezpečnosti silniční infrastruktury*. In: . Brusel: EVROPSKÝ PARLAMENT A RADA EVROPSKÉ UNIE, ročník 2008, číslo 96.

3 *OWASP* [online]. USA: OWASP, 2001 [cit. 2016-03-21]. Dostupné z: http://www.owasp.org

4 WEBER, Filip. Penetrační testy v bezpečnostní analýze informačního systému. In: *Svět sítí*[online]. 2007 [cit. 2016-05-04]. Dostupné z: http://www.svetsiti.cz/clanek.asp?cid=Penetracni-testy-v-bezpecnostni-analyze-informacniho-systemu-28102007

5 PŘIBYL, Tomáš. Penetrační testování webových služeb. In: Nezávislý odborný on-line magazín ICT SECURITY [online]. 2014 [cit. 2016-05-04]. Dostupné z: http://www.ictsecurity.cz/odborne-clanky/penetracni-testovani-webovych-sluzeb.html

6 ZÁVODSKÝ, Petr (ed.). Úctyhodný Open Web Application Security Project (OWASP). *Root.cz* [online]. 2015, 2015, 1 [cit. 2016-03-22]. Dostupné z: http://www.root.cz/clanky/uctyhodny-open-web-application-security-project-owasp/

7 ZÁVODSKÝ, Petr (ed.). Úctyhodný Open Web Application Security Project (OWASP). *Root.cz* [online]. 2015, 2015, 1 [cit. 2016-03-22]. Dostupné z: http://www.root.cz/clanky/uctyhodny-open-web-application-security-project-owasp/

8 *OWASP TOP 10 - 2013: Deset nejkritičtějších bezpečnostních rizik webových aplikací* [online]. In: . s. 22 [cit. 2016-03-23]. Dostupné z: https://www.owasp.org/images/f/f3/OWASP_Top_10_-_2013_Final_-_Czech_V1.1.pdf

9 *OWASP TOP 10 - 2013: Deset nejkritičtějších bezpečnostních rizik webových aplikací* [online]. In: . s. 22 [cit. 2016-03-23]. Dostupné z: https://www.owasp.org/images/f/f3/OWASP_Top_10_-_2013_Final_-_Czech_V1.1.pdf

10 PHP.net. *PHP.net* [online]. 2001 [cit. 2016-03-28]. Dostupné z: http://php.net

11 *OWASP TOP 10 - 2013: Deset nejkritičtějších bezpečnostních rizik webových aplikací* [online]. In: . s. 22 [cit. 2016-03-23]. Dostupné z: https://www.owasp.org/images/f/f3/OWASP_Top_10_-_2013_Final_-_Czech_V1.1.pdf

12 OWASP Enterprise Security API. *OWASP* [online]. 2007 [cit. 2016-03-28]. Dostupné z: https://www.owasp.org/index.php/ESAPI

13 OWASP Application Security Verification Standard Project. *Owasp.net* [online]. [cit. 2016-04-04]. Dostupné z: https://www.owasp.org/index.php/ASVS

14 ESAPI Authenticator and User APIs. *Owasp-esapi* [online]. [cit. 2016-04-04]. Dostupné z: http://owasp-esapi-java.googlecode.com/svn/trunk_doc/latest/org/owasp/esapi/Authenticator.html

15 JavaScript Tutorial. *W3schools.com* [online]. [cit. 2016-04-25]. Dostupné z: http://www.w3schools.com/js/

16 CO JE TO PHISHING. *Hoax* [online]. [cit. 2016-04-25]. Dostupné z: http://www.hoax.cz/phishing/co-je-to-phishing

17 XSS (Cross Site Scripting) Prevention Cheat Sheet. *Owasp.org* [online]. [cit. 2016-04-04]. Dostupné z: https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet

18 PHP htmlspecialchars() Function. *W3schools* [online]. [cit. 2016-04-04]. Dostupné z: http://www.w3schools.com/php/func_string_htmlspecialchars.asp

19 PEJŠA, Jan. Co je Cross-Site Request Forgery a jak se mu bránit. In: Zdroják.cz [online]. 2008 [cit. 2016-05-04]. Dostupné z:

https://www.zdrojak.cz/clanky/co-je-cross-site-request-forgery-a-jak-se-branit/

20 *OWASP Testing Guide v4* [online]. In: . s. 222 [cit. 2016-04-24]. Dostupné z: https://www.owasp.org/images/5/52/OWASP_Testing_Guide_v4.pdf

21 HTML5: A vocabulary and associated APIs for HTML and XHTML. *W3.org* [online]. 2014 [cit. 2016-04-25]. Dostupné z: https://www.w3.org/TR/html5/

22 Apache HTTP Project. *Apache HTTP Project* [online]. [cit. 2016-04-25]. Dostupné z: https://httpd.apache.org/

23 JQuery. *JQuery* [online]. [cit. 2016-04-25]. Dostupné z: https://jquery.com/

24 PHP. *PHP: Hypertext preprocessor* [online]. [cit. 2016-04-25]. Dostupné z: http://php.net/

25 About MariaDB. *MariaDB foundation* [online]. [cit. 2016-04-25]. Dostupné z: https://mariadb.org/about/

26 PhpMyAdmin. *PhpMyAdmin* [online]. [cit. 2016-04-25]. Dostupné z: https://www.phpmyadmin.net/

27 Htop - an interactive process viewer for Unix. *Hisham.hm* [online]. [cit. 2016-04-25]. Dostupné z: http://hisham.hm/htop/

28 MPDF. *MPDF* [online]. [cit. 2016-04-25]. Dostupné z: http://www.mpdf1.com/mpdf/index.php

29 PHPExcel. *CodePlex.com - Project Hosting for Open Source Software* [online]. [cit. 2016-04-25]. Dostupné z: https://phpexcel.codeplex.com/

30 OWASP TOP 10 - 2013: Deset nejkritičtějších bezpečnostních rizik webových aplikací [online]. In: . s. 22 [cit. 2016-03-23]. Dostupné z: https://www.owasp.org/images/f/f3/OWASP_Top_10_-_2013_Final_-_Czech_V1.1.pdf

31 *OWASP Testing Guide v4* [online]. In: . s. 222 [cit. 2016-04-24]. Dostupné z: https://www.owasp.org/images/5/52/OWASP_Testing_Guide_v4.pdf

32 Seznamte se s Mozillou. *Mozzilla* [online]. 2016 [cit. 2016-05-01]. Dostupné z: https://www.mozilla.org/cs/about/

33 What is Java? *Java* [online]. 2016 [cit. 2016-05-01]. Dostupné z: http://www.java.com/en/download/whatis_java.jsp

34 OWASP Zed Attack Proxy Project. *Owasp.org* [online]. 2016 [cit. 2016-05-01]. Dostupné z: https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project

35 Wireshark. *Wireshark* [online]. 2016 [cit. 2016-05-01]. Dostupné z: https://www.wireshark.org/

36 Doplňky Mozzilla - Live HTTP Headers. *Addons.mozilla.org* [online]. 2016 [cit. 2016-05-01]. Dostupné z: https://addons.mozilla.org/cs/firefox/addon/live-http-headers/

37 Doplňky Mozzilla - Firebug. *Addons.mozilla.org* [online]. 2016 [cit. 2016-05-01]. Dostupné z: https://addons.mozilla.org/cs/firefox/addon/firebug/

38 Doplňky Mozzilla - Tamper Data. *Addons.mozilla.org* [online]. 2016 [cit. 2016-05-01]. Dostupné z: https://addons.mozilla.org/cs/firefox/addon/tamper-data/?src=search

39 *JetBrains - PhpStorm* [online]. 2016 [cit. 2016-05-01]. Dostupné z: https://www.jetbrains.com/phpstorm/

40 Print_r. *Php.net* [online]. 2016 [cit. 2016-05-01]. Dostupné z: http://php.net/manual/en/function.print-r.php

41 PHP Data Objects. *Php.net* [online]. 2016 [cit. 2016-05-01]. Dostupné z: http://php.net/manual/en/book.pdo.php

42 Ldap_bind. *Php.net* [online]. 2016 [cit. 2016-05-01]. Dostupné z: http://php.net/manual/en/function.ldap-bind.php

43 DVOŘÁK, Tomáš. X-FRAME-OPTIONS hlavička. In: *TOMAS DVORAK programming, travel, sport* [online]. 2012 [cit. 2016-05-04]. Dostupné z: http://www.tomas-dvorak.cz/posts/x-frame-options-hlavicka/

44 Understanding SHA-1 Vulnerabilities — Is SSL No Longer Secure? *ENTRUST* [online]. 2016 [cit. 2016-05-01]. Dostupné z: https://www.entrust.com/understanding-sha-1-vulnerabilities-ssl-longer-secure/

45 Runtime Configuration. *Php.net* [online]. 2016 [cit. 2016-05-01]. Dostupné z: http://php.net/manual/en/session.configuration.php

46 All about SHA1, SHA2 and SHA256 hash algorithms. *TBS Internet* [online]. 2016 [cit. 2016-05-01]. Dostupné z: https://www.tbs-certificates.co.uk/FAQ/en/sha256.html

47 Cross-site Scripting (XSS). *Owasp* [online]. 2016 [cit. 2016-05-04]. Dostupné z: https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)

48 HTML Symbols: HTML Symbol Entities. *W3schools* [online]. 2016 [cit. 2016-05-01]. Dostupné z: http://www.w3schools.com/html/html_symbols.asp

49 Doplňky Mozzilla - Tamper Data. *Addons.mozilla.org* [online]. 2016 [cit. 2016-05-01]. Dostupné z: https://addons.mozilla.org/cs/firefox/addon/tamper-data/?src=search

50 WALSH, David. Image Data URIs with PHP. In: *DWB* [online]. 2012 [cit. 2016-05-04]. Dostupné z: https://davidwalsh.name/data-uri-php

51 Data URIs. *MDN* [online]. 2016 [cit. 2016-05-01]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/data_URIs

9. Seznam obrázků

Obrázek 1 - Logo organizace OWASP (Zdroj: 3) 7

Obrázek 2 – Autoři aplikace (Zdroj: Autor) 23

Obrázek 3 - Zanášecí rozhraní (Zdroj: Autor) 26

Obrázek 4 - Položka ve výpisu pro operátora (kodéra) (Zdroj: Autor) 27

Obrázek 5 - Rozcestník pro uživatele ŘSD (Zdroj: Autor) 28

Obrázek 6 - Rozhraní výpisu s oknem stav řešení (Zdroj: Autor)..... 29

Obrázek 7 - Průběh záznamu reakcí (WorkFlow) (Zdroj: Autor) 29

Obrázek 8 - Výběr podsystému (Zdroj: Autor)..... 30

Obrázek 9 - Ukázka kombinace JS s AJAXem hledající historii reakcí k deficitu (Zdroj: Autor)..... 31

Obrázek 10 - Schéma původní verze databáze pro aplikaci TEN-T (Zdroj: Autor) 34

Obrázek 11 - Zjednodušené schéma aplikace (Zdroj: Autor) 35

Obrázek 12 - Schéma společné databáze systému CEBASS (Zdroj: Autor) 36

Obrázek 13 – Schéma současné aplikace Cebass (Zdroj: Autor) 37

Obrázek 14 - Přihlašovací dialog (Zdroj: Autor) 39

Obrázek 15 - Hlavička http požadavku s daty typu POST (Zdroj: Autor) 40

Obrázek 16 - PHP kód s SQL dotazem na výběr uživatele z DB (Zdroj: Autor) .. 40

Obrázek 17 - Podmínka ověřující vstupy (Zdroj: Autor) 41

Obrázek 18 - Implementace ochrany LDAP před útokem hrubou silou (Zdroj: autor)..... 43

Obrázek 19 - Nastavení hlaviček pomocí PHP 43

Obrázek 20 - Ukázka inicializace některých proměnných sezení (Zdroj: Autor) 44

Obrázek 21 - Ukázka šifrovaného spojení vůči webovému serveru (Zdroj: Autor) 45

Obrázek 22 -Implementace opravy, ukončení sezení po určitém čase (Zdroj: Autor) 46

Obrázek 23 - Implementace pokročilé kontroly relace uživatele (Zdroj: Autor) 47

Obrázek 24 - Náhled na tabulku se systémy (Zdroj: Autor) 48

Obrázek 25 - Ukázka implementace kontroly pracovního adresáře (Zdroj: Autor) 48

Obrázek 26 - Ukázka všech možných rozhraní pro vkládání reakcí (Zdroj: Autor) 49

Obrázek 27 - Vkládání reakce v rozhraní detailu (Zdroj: Autor) 50

Obrázek 28 - Následek úspěšného útoku (Zdroj: autor) 50

Obrázek 29 - Ukázka použití doplňku Tamper Data, který dokáže měnit obsah dat posílaných přes POST (Zdroj: Autor) 51

Obrázek 30 -Implementace funkce htmlspecialchars() před uložením do DB (Zdroj: Autor) 52

Obrázek 31 - Oprava na straně výpisu dat z DB (Zdroj: Autor) 52

Obrázek 32 - Nastavení pro složku upload (Zdroj: Autor) 53

Obrázek 33 - Příklad, kdy parametr Options obsahuje hodnotu Indexes (Zdroj: Autor) 53

Obrázek 34 - Původní funkce na výpis fotky jako dat (Zdroj: 50) 54

Obrázek 35 - Vlastní implementace funkce převádějící obrázek na data (Zdroj: Autor) 55

Obrázek 36 - Příklad volání funkce z rozhraní exportu do PDF (Zdroj: Autor) . 55

10. Seznam tabulek

Tabulka 1 - Přehled kategorií 2010 a 2013 (Zdroj: 9) 8

Tabulka 2 - Tabulka uživatelů (Zdroj: Autor) 9

Tabulka 3 - Příklad tabulky vztahů při nepřímé referenci (Zdroj: Autor) 15