Vysoká škola ekonomická v Praze
Fakulta informatiky a statistiky
Vyšší odborná škola informačních služeb v Praze

Daniel Císař

Bezpečnost webových aplikací (PHP) implementace zabezpečení

Bakalářská práce

2007

zadávací list

Prohlašuji, že jsem bakalářskou práci na téma Bezpečnost webových aplikací (PHP) - implementace zabezpečení zpracoval samostatně a použil pouze zdrojů, které cituji a uvádím v seznamu použité literatury.

V Praze dne: *Podpis:*

Obsah

Úvod

Celosvětový růst požadavků na webové aplikace v posledních letech přinesl mnoho moderních technologií. Technologií, které se snaží vyjít uživatelům co nejvíce vstříc a usnadnit jim komunikaci a rozvoj. Požadavky stoupají nejen na funkčnost a estetiku resp. design, ale hlavně na bezpečnost.

Internet jako takový od začátku byl koncipován jako nezabezpečený tok dat. Pro svoji jednoduchost se záhy rozšířil i pro komerční využití mezi nestátní subjekty. Protokoly, které zajišťovali komunikaci po síti nebyly navrženy jako bezpečné, a proto se přesunula odpovědnost za implementaci zabezpečení na stranu programátorů, tedy do vrstvy aplikační.

Už mnoho příkladů dokázalo, že se nevyplatí nevěnovat zvýšenou pozornost bezpečnostní politice organizace, potažmo aplikací, které využívá. V honbě za novými verzemi programů a včasnému zprovoznění aplikací, se často zapomíná na čisté a hlavně bezpečné programování.

Záměrem mé diplomové práce je implementace vhodných funkcí, které zajistí zvýšenou bezpečnost dle běžných standardů. Ta je nárokována u produktů typu e-shop a jiných. aplikací, které pracují s citlivými údaji zákazníků (uživatelů systému) nebo zprostředkovávají ekonomické přesuny.

Toto řešení, nebo návod chcete-li, si bere za cíl seznámit čtenáře s běžnými útoky na webové aplikace. Je tedy vhodné pro projekty, které jsou již funkční a autor projektu nevěnoval bezpečnostní otázce příliš času a energie. Tato práce popisuje opravu a implementaci lepšího zabezpečení. Pro pokročilejšího programátora by bylo vhodnější sáhnout po již hotovém produktu aplikace typu open source framework, kterých je na internetu na výběr dost.

To ale neznamená, že by tato práce nebyla přínosná. Obzvláště pro menší projekty a pro programátory, kteří nechtějí sázet na cizí framework aplikace a chtějí si naprogramovat nějaký vlastní, přichází vhod souhrn běžných bezpečnostních rizik webových aplikací a jejich řešení.

Jak jsem zmínil, technologie se rychle vyvíjejí, a s tím jde ruku v ruce i vývoj techniky útoků na webové aplikace. Je tedy více než pravděpodobné, že se během doby objeví útoky nové, a i proto v této práci na čtenáře (nejen programátory) apeluji, aby nezanedbali sebevzdělávání v oblasti bezpečného programování a sledovali neustále různé informační kanály.

Kapitola 2

Projekt e-shopu

2.1 Funkční specifikace

Tento projekt byl tvořen od začátku na zakázku a nebylo zamýšleno další využití napsaného kódu. Postupně však objednatel vršil požadavky až se z toho stala plnohodnotná webová aplikace s administračním rozhraním. Jelikož se situace, které řeší tento kód v jiných obchodech opakují, je možno nasadit tento projekt na další menší projekty internetového obchodování.

Bohužel nevýhoda však spočívá v nutnosti většího zásahu do kódu v případě jiného grafickému rozložení (layoutu). Tento projekt od začátku nepočítal se šablonovacími systémy[1], které jsou v PHP dostupné a které takový přechod umožňují a usnadňují. Jelikož by tato implementace nijak zásadně neovlivnila lepší zabezpečení aplikace, rozhodl jsem se ponechat současný fungující systém.

Jednotlivou logiku vykreslování částí kódu zajišťují funkce. Každý skript, který je jakýmsi modulem nebo komponentou nese stejnou hlavičku a patičku. Tato hlavička a patička se připojí k jednotlivým modulům pomocí funkce *require()*. V hlavičce se řeší společné věci pro všechny skripty, které mají na starost zobrazení frontendu obchodu. Mezi hlavní záležitosti, které hlavička definuje a spravuje patří: připojení k databázi, správa session, přihlášení/odhlášení uživatele, pokud je uživatel přihlášený - kontroluje se, zda mu nevypršela session, dále pak změna měny (CZK/SKK).

Funkce, které zajišťují zobrazení výstupu nebo spravují autentizaci uživatele jsou na jednom místě v souboru *fce.php*. Takto se zajistí lepší správa a příp. opravy v php nebo generovaném html kódu. S tím se samozřejmě zvyšuje i znovupoužitelnost kódu. Tento soubor spolu s hlavičkou (*./prip/heade.php*) a patičkou (*.prip/footer.php*) také doznal největších změn, co se bezpečnostních opatření týče.

1 http://php.vrana.cz/sablony.php

2.2 Struktura aplikace

Aplikace obsahuje několik částí které se starají o výstup. Uvedu zde krátký popis komponent, které e-shop obsahuje:

adm.php

- změna nastavení údajů o uživateli, změna hesla, emailu, možné zobrazit pouze po přihlášení.

do_kosiku.php

- komponenta založí session, pokud nebyla již založena a vytvoří proměnnou, do které se ukládá id, velikost a množství nakoupeného zboží.

fce.php

- funkce, které jsou nutné pro běh aplikace

index.php, nahled_kat.php, nahled_subkat.php, nahled_zbo.php

- hlavní stránka, vybraná hlavní kategorie, vybraná podkategorie, vybrané zboží. Všechny tyto stránky zobrazují zboží, které je buď v akci (*index.php*), v dané kategorii (*nahled_kat.php*), v dané subkategorii nebo zobrazujeme jednotlivé zboží (*nahled_zbo.php*). V subkategoriích můžeme vyhledávat dle značky a velikosti. Obchod funguje pouze s dvěma podkategoriemi, ale dá se lehkými úpravami kódu přepsat na více kategorií.

orders.php

- objednávky uživatele, možné zobrazit pouze po přihlášení.

registrace.php

- registrace nového uživatele – zde nejvyšší riziko nezabezpečených vstupů, jelikož zde má případný útočník nejvíce možností přes registrační formulář

zaplatit.php

- zde se po potvrzení zobrazeného košíku (viz *zobraz_kosik.php)* vypíšou veškeré údaje nutné k objednávce: pro zkontrolování adresa, nacionály uživatele a objednávka samotná. Až do této fáze je možné měnit měnu z CZK na SKK a obráceně. Po nutném přihlášení je možné objednávku definitivně potvrdit a poslat. Pokud vše dopadne dobře, je zaslán email jak zákazníkovi, tak správcovi aplikace, že objednávka byla přijata.

zapomenute_heslo.php

- pokud uživatel zapomene heslo, pošleme mu nové vygenerované heslo na email, který vyplnil při registraci. Uživatel je vyzván k bezodkladnému přihlášení a musí to stihnout do daného limitu, jinak se mu účet zablokuje

znacky.php

- zobrazujeme zboží od jednotlivých značek

zobraz.php

- zde se načítají textové informace o obchodu: jak nakupovat, kontakty atp.

zobraz_kosik.php

- pokud uživatel vložil nějaké zboží do virtuálního košíku, zde ho zobrazíme. Může zde také měnit množství jednotlivého zboží a také jednotlivé zboží z košíku mazat.

. /prip/session_handler.php

- vlastní session handler, více v kapitole 5

./prip/header.php

- společná hlavička, která se načítá ve většině komponent

.prip/footer.php

- společná patička, která se načítá ve většině komponent

Administrační část

.admin/admin.php

- tento skript obsahuje metodou dispatch veškeré moduly, které se pomocí funkce *include* a parametru *$_GET["ak"]* přepínají mezi jednotlivými možnostmi.

```
<?php
//jsme prihlaseni
if ($jePrihlasen){
                switch($_GET["ak"]){
                    case "adm":
                        require_once("fce/adm.php");
                    break;

                   case "menu":
                            require_once ("fce/menu.php");
                   break;
                   //zkraceno
                }
 }
?>
```

.admin/index.php

– stránka pro zobrazení přihlašovacího formuláře, veškeré další stránky obsahují také

formulář a pokud není uživatel přihlášen nebo mu vypršela session, je mu zobrazen pouze tento formulář

.admin/fce/

- zde se nalézají veškeré moduly: správa údajů administrátora (*adm.php*), vymazání zboží (*dzbozi.php*), editace zboží (*ezbozi.php*), správa menu (*menu.php*), novinek (*news.php*), přidání nového zboží (*nzbozi.php*), správa objednávek (*orders.php*), správa dealerských slev (*slevyd.php*), velikostí (*velikost.php*), správa uživatelů obchodu (*zak.php*), správa zboží (na skladě/všechno) (*zbozi.php*) a správa značek (*znacka.php*)

grafika

- veškerá obrázky zboží se ukládají do adresáře zboží ./*zbozi*/, grafika samotného webu do adresáře ./*img*/

Veškeré skripty jsou přiložené na CD v adresáři ./*e-shop*.

2.3 Požadavky na aplikaci

Následující seznam jasně definuje požadavky na funkčnost jak z pohledu obchodníka, tak z pohledu zákazníka, který si chce objednat zboží. Funkce jsou řazeny od fundamentálních až po ty méně důležité:

- ✔ ochrana osobních údajů zákazníka
- ✔ vlastní administrační rozhraní
- ✔ bezproblémová registrace
- ✔ přehledná struktura obchodu
- ✔ prohlížení zboží bez nutnosti přihlášení nebo registrace
- ✔ realizace nákupu pomocí nákupního košíku
- ✔ nákup v cizí měně (Slovenské koruny)
- ✔ informování o novinkách

2.4 Použité technologie

2.4.1 PHP

PHP je Open Source serverový (server-side) skriptovací jazyk. Byl od začátku navržen pro webové aplikace, které dynamicky generují obsah. Server, na kterém je PHP nainstalováno, za běhu zpracovává požadavky a provádí interpretaci jednotlivých skriptů. Veškerý výsledek následně zašle klientovi. Většinou se jedná o výstup (X)HTML dokumentů nebo jiných výstupů (XML, PDF, renderované grafické prvky, atd.). PHP není nijak závislé na určité platformě, proto je nasazováno v různých modifikací na velké množství serverů a jeho rozšířenost a obliba neustále roste [1].

2.4.2 MySQL

Rychlá a robustní relační databáze, která splňuje požadavky současných webových aplikací. Je to víceuživatelský a vícevláknový (multi-threaded) server, který používá SQL syntaxi. Jedná se o celosvětově nejoblíbenější open source databázi. Tato databáze je k dispozici pod dvěma různými licenčními podmínkami. Jestliže hodláme dodržovat podmínky licence GPL, můžeme použít verzi *Community Edition*[2]. Pokud pro nás není možné použít GPL licenci, je tu možnost koupit si licenci komerční *MySQL Enterprise* [1].

2.4.3 Nástroje na vývoj aplikace

Neméně důležitým aspektem vývoje bezpečného softwaru je dobré vývojové prostředí. Jedná se o tzv. *IDE (Integrated development environment). IDE* se většinou skládá z: editoru, kompileru a/nebo interpreteru, automatizačních nástrojů a většinou obsahuje i důležitý debugger. Lepší *IDE* mají správce verzí kódu, prohlížeč tříd a funkcí, inspektor objektů, hierarchický diagram tříd, nástroje na návrh uživatelského grafického rozhraní (*GUI*) aj. pomůcky nejen pro objektové programování (*OOP*).

Při své práci jsem použil všestranný open source *IDE Eclipse*. Jeho rozšiřitelnost vzala za vděk i komunita PHP programátorů a pro velkou základnu programátorů je vyvíjen speciální plugin. Celý projekt se jmenuje *PDT* (*PHP Development Tools*)[3]. Společně se Zend Debuggerem[4] tvoří silnou dvojici pro vývoj bezpečných přehledných kódů webových aplikací. Nyní se blíží vydání

2 http://dev.mysql.com/
3 http://www.eclipse.org/pdt/
4 http://www.zend.com/pdt

verze 1.0, která bude kompletní s podporou Debuggování – krokování, sledování hodnot proměnných a jejich změna, tzv. *breakpoints*[5] i s podmínkami zastavení. Bez těchto funkcí se velké aplikace dělají velice těžko.

Další ceněnou distribucí všeho v jednom je distribuce *XAMPP*[6] (*Apache*, *MySQL*, *PHP*, *Perl*). Je určena pro nejen pro operační systém Windows, ale i pro Linuxové distribuce a dokonce i pro Mac OS X a Solaris. V tomto balíku je toho ještě více než jen zmiňovaná čtveřice a vše jednoduše nainstalované pomocí instalátoru. Správa a diagnostika jednotlivých částí balíku je přes jednoduché webové rozhraní.

5 Body kde má aplikace zastavit interpretaci kódu
6 http://www.apachefriends.org/en/xampp.html

Kapitola 3

Osvětové projekty - bezpečnostní otázky programování

V době velkého rozvoje open source webových aplikací, které se těší stále větší oblibě, rostou i požadavky na jejich bezpečnost. Stále více lidí používá open source a nevidí v tom pouze alternativu, ale také úsporu peněz. Výhoda takového software je právě jeho počáteční cena a otevřený kód, který za jistých podmínek můžeme dále použít. Bohužel to s sebou nese daleko více problémů, než u kódu uzavřeného.

Tento způsob vývoje a distribuce software klade vysoké nároky na základní robustní architekturu aplikace. Kvalitní projekty mohou sloužit edukativně jak pro začínající, tak i pro pokročilé programátory. Pokud je aplikace dosti oblíbená, najde se spoustu lidí, kteří věnují čas na zlepšení a opravu kódu. Avšak nekvalitní projekty mohou být špatným příkladem, pokud na to nikdo neupozorní. Každý den spousta odborníků upozorňuje na možné bezpečnostní díry v těchto aplikacích. Informace se však objevují na různých místech a ztrácí tak na své hodnotě.

Naštěstí existují i organizace, které fungují na principech open source a fungují jako zpětná vazba pro open source komunitu. Tyto organizace žijí ze sponzorských darů velkých softwarových společností, kterým vlastně svojí činností pomáhají rozvíjet vzdělání programátorů a snižovat tak rizika úniku tolik ceněných informací. Otevřenost kódu je tedy v tom smyslu výhodou.

3.1 OWASP

The Open Web Application Security Project (*OWASP*) je organizace resp. otevřená komunita, která se projekuje současný stav bezpečnosti webového aplikačního software. Díky agregovanému informování, napomáhá důvěryhodnosti open source aplikací. Nabízí užitečné informace a nástroje, které jsou jsou dostupné všem, kdo chtějí přispět k efektivnímu boji za vylepšení zabezpečení webových aplikací. Obhajují zlepšení aplikační bezpečnosti jako lidský, procesní a technologický problém. Nejvíce efektivní postoj jak získat lepší výsledky, je zabývat se jimi jako dílčími oblastmi, které je všechny třeba neustále vylepšovat. Píší [2], že jejich nezávislost na komerčním

tlaku jim dovoluje nabízet nezaujaté, praktické a ekonomicky efektivní informace o bezpečnosti aplikací. Nejsou spojeny s žádnou technologickou společností, ačkoliv také nabízejí podporu užití bezpečnostních technologií na komerční bázi. Dále píší, že jsou nevýdělečnou společností, což jim zaručuje dlouhodobý rozvoj projektu.

3.1.1 OWASP Top Ten

OWASP Top Ten projekt nabízí obecný konsenzus mnoha odborníků, kteří se zabývají bezpečnostní politikou vývoje software. Zahrnuje tak návrháře, vývojáře i správce, aby pokryl co nejširší možné spektrum otázek kolem bezpečného programování.

Tento projekt poukazuje, že stále spousta vývojářských firem nedbá dostatečně na bezpečnost svých aplikací a tak se členové organizace OWASP snaží zlepšit situaci dostatečným informováním. Jedná se o elementární informace, které je třeba využít při vývoji software.

Hlavním cílem projektu je vytvořit dokument, který shrnuje nejčastěji se objevované chyby a nabízí způsoby, jak se jich vyvarovat. Dokument se vytváří postupně časem, kdy probíhají výměny názoru jednotlivých zainteresovaných odborníků a široké veřejnosti v diskuzních příspěvcích[7]. Dokument se jednotlivými připomínkami mění tak dlouho, dokud ho odborníci neschválí jako stabilní.

Tato práce se opírá o verzi *OWASP Top Ten 2004* s následujícím žebříčkem [3]:

- nezvalidované vstupy
- přístupová práva
- autentizace a správa sessions
- cross site scripting (*XSS*)
- přetečení paměti (buffer overflows)
- *injection*
- Vypisování chyb (errors)
- nezabezpečené uložení dat
- denial of service (*DoS*)
- Nesprávné nastavení

7 https://lists.owasp.org/mailman/listinfo/owasp-topten

Důkazem měnících se požadavků na bezpečnost a vývoje programování vůbec, je nový stabilní dokument *OWASP Top Ten* 2007, který mění pozici jednotlivých útoků. Aktuální dokument *OWASP Top Ten 2007* popisuje tyto nejčastější chyby:

- cross site scripting (*XSS*)
- *injection*
- exekuce nebezpečného souboru
- nezabezpečený přímý odkaz na objekt
- Cross Site Request Forgery (*CSRF*)
- únik informací a nesprávná správa chyb
- špatná správa *sessions* a autentizace
- nešifrování citlivých dat
- nezabezpečená komunikace
- přístup k údajům přes URL adresu

3.1.2 OWASP PHP Project

Mnohem důležitější informační centrála pro vývoj webových aplikací v PHP je však projekt, který se specializuje na tento jazyk. Jsou zde [4] nastíněná konkretní řešení i s kódem, který je třeba integrovat do aplikací.

3.2 PHP Security Consortium

Dalším zdrojem informací je PHP Security Consortium[5]. Pro začátek je dobré si přečíst a osvojit myšlenky v dokumentu *PHP Security* Guide. Bohužel již neaktualizovaný dokument stačí pouze na seznámení s tématem. Jiné projekty jako *SecurityFocus Summaries* a *PhpSecInfo* také nebyly dlouho aktualizovány.

PhpSecInfo je však nadále dobrým nástrojem pro administrátory serverů, na kterých běží PHP. Tato aplikace reportuje informace o prostředí, ve kterém PHP běží. Detekuje známe problémové nastavení běhového prostředí a nabízí rady jak toto nastavení zlepšit, s cílem dosažení lepšího dílčího zabezpečení serveru.

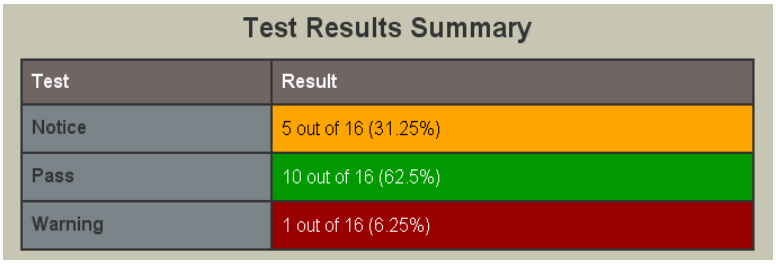
Kapitola 4

Detekce ohroženého kódu

Po delším průzkumu jsem musel konstatovat, že v PHP neexistuje v současné době mnoho nástrojů, které by pomáhali programátorům při odhalování bezpečnostních chyb. Nakonec sebevzdělávání v této oblasti vždy ponese programátor na svých bedrech. Pojďme se tedy na to málo, co nám jednotlivé projekty nabízí, podívat.

4.1 Detekce nevhodného nastavení serveru

Mezi tuto kategorii bezesporu patří nástroj PHPSecInfo od organizace *PHP Security* Consortium. Vedoucí tohoto projektu je Ed Finkler. Tento nástroj provádí testy konfiguračního souboru (php.ini) a vypíše tabulku s jednotlivými doporučeními, jak příp. nedostatky v nastavení serveru upravit. Provedl jsem takovou analýzu na produkčním serveru e-shopu a získal následující výsledek:



Test Results Summary

Test	Result
Notice	5 out of 16 (31.25%)
Pass	10 out of 16 (62.5%)
Warning	1 out of 16 (6.25%)

Jedno varování bylo na nastavení: allow_url_fopen="on". Na stránkách pak anglicky podávají vysvětlení: „*Když je toto nastavení povoleno, je povoleno PHP funkcím jako file_get_contents() a příkazům include nebo requiere získávat data ze vzdálených zdrojů, jako např. z FTP nebo webových stránek. Programátoři často na toto zapomínají a neprovádějí správně filtrování vstupů, když posílají uživatelem poskytnutá data těmto funkcím. Což otevírá tyto funkce snadno k vložení nebezpečného kódu útočníkem. Velké množství vložení nebezpečného kódu, které jsou v PHP webových aplikací reportovány jsou zapříčiněny právě kombinací nastavení allow_url_fopen="on" a nesprávného filtrování vstupů.*"

K filtrování dat od uživatele se dostaneme v části 5.1 Ošetření vstupů a výstupů aplikace. Můžu však s klidným svědomím říci, že můj projekt je této bezpečnostní chyby prost. Veškeré *include*, *require* příkazy jsou napevno a nevstupují zde žádné uživatelem stanovené hodnoty. Pokud

však tomu tak je, je třeba filtrovat tyto vstupy či jednoznačně určit (např. pomocí pole povolených hodnot), jaké soubory (adresy) mohou být do skriptu nahrány.

4.2 Aplikační nástroje detekce ohroženého kódu

Do této kategorie se plně dají počítat open source projekt OWASP WebScarab[8] a komerční velice drahý produkt Acunetix Web Vulnerability Scanner[9] nebo jiné podobné nástroje. Osobně bych sázel na prudký rozvoj projektu OWASP, jelikož aplikace od Acunetix je příliš drahá a pouze vypisuje jednotlivé chyby, které jsou v databázi firmy Acunetix.

Není zde tedy vůbec pozitivní účinek na sebevzdělávání programátora, který tak zleniví a spoléhá se pouze na jednu aplikaci. V použití aplikace WebScarab zase vidím nevýhodu opačného rázu. Jelikož zatím není dostupný kompletní manuál, jak tento nástroj použít, je tu riziko špatného použití tohoto nástroje a chybného úsudku o bezchybnosti testované webové aplikace.

Je tedy třeba vše zdokumentovat a zpřístupnit možnosti testování webových aplikací širokým masám programátorů PHP (aj. programovacích jazyků). V této oblasti je velký příslib [6] projekt OWASP *WebScarab NG* (next generation). Slibují tu lepší rozhraní, které by mělo být více uživatelsky přívětivější.

4.3 Ruční oprava kódu

Co tedy zbývá programátorům, když dostupné nástroje buď nejsou kompletní nebo jsou ekonomicky nedosažitelné. Zbývá jediné, projít kód řádek od řádku. Pomoci mohou, jak jsem již zmínil, nástroje pro vývoj (*IDE*) a jejich debuggovací rozhraní. Dbát by měl programátor na neustálém sebevzdělávání a držení kroku s vývojem.

Vývoj technologií je totiž velice rychlý a techniky, jak se bránit před příp. útočníky, které fungovali dříve, dnes již fungovat nemusejí, anebo existují mnohem účinnější řešení bezpečnostního problému. Bohužel je to neustálý boj a většinou jsou útočníci napřed. Mají také časovou výhodu a bohužel i pro ně jsou k dispozici nedokumentované nástroje, které pomáhají objevit bezpečnostní mezery. Nedávno např. unikl AJAXový nástroj pro detekci *XSS* a *SQL injection – Jikto* [7]. Tyto doslova zbraně v rukou útočníka mohou napáchat nezměrné dalekosáhlé škody.

Pomoci mohou výše uvedené osvětové projekty a jejich publikace (OWASP Guide 2.0, PHP Security Guide) a další zdroje. Čím více relevantních zdrojů, tím lépe. Další projekt organizace *OWASP Web Goat Project* [8], se také snaží pomocí školení přispět k bezpečnějším aplikacím. Veškeré školení probíhá ve webové aplikaci postavené na platformě *Java*. Základy a tech-

8 http://www.owasp.org/index.php/Category:OWASP_WebScarab_Project
9 http://www.acunetix.com/vulnerability-scanner/

niky útoků se v jednotlivých jazycích příliš nemění, proto je tento projekt prospěšný pro většinu programátorů webových aplikací.

4.4 Bezpečnostní chyby v projektu

Největší problém bylo zabezpečení proti *CSRF* (viz níže), jelikož celá administrační část byla touto chybou postižena. Bylo třeba tedy přepracovat toto rozhraní, aby se tak minimalizovala hrozba CSRF útoku.

Bylo potřeba zkontrolovat, zda-li se všude testuje očekávaná hodnota např. pomocí funkce *is_numeric()* aj., nebo zda-li vyhovuje regulárnímu výrazu. Tento krok spolu s funkcí *mysqli_real_escape_string()* spolehlivě uzavřel dveře SQL injection útokům.

Logické bezpečnostní díry byly např. ve formuláři pro registraci a úpravu osobních údajů. Často aplikace testuje, zda-li uživatel příslušné pole vyplnil, ale zapomíná se na odstranění mezer např. funkcí *trim()*. V neopravené verzi totiž mohl uživatel vyplnit do textového pole pouze mezeru.

Musel se také opravit kód, který mohl příp. útočníkovi pomoci získat nové platné heslo. Pokud je v aplikaci možnost zaslání zapomenutého hesla, je nutné při změně emailu vyžadovat potvrzení změny platným heslem. Jinak by si mohl někdo, kdo má fyzický přístup k počítači přihlášeného uživatele změnit email na svůj email a poté si nechat zaslat heslo.

Přepracoval jsem správu session, celá správa session se přesunula do databáze, kvůli hrozbě zcizení session.

V autentizaci jsem zvýšil bezpečnost přidáním kontroly hlavičky *User-Agent* a provedl její celkovou revizi.

Opravil jsem také neinicializované proměnné, není to sice bezpečnostní oprava, ale je dobrým zvykem proměnné inicializovat. Také je potřeba zmínit, že celý projekt se opírá o standardní nastavení PHP serveru „*register_globals=off*“, tedy není možné přistoupit k proměnným z požadavků jinak, než přes superglobální proměnné.

Kapitola 5

Řešení problémových částí kódu

5.1 Ošetření vstupů a výstupů aplikace

S lehkou nadsázkou se dá tvrdit, že základ bezpečnosti aplikace stojí a padá na ošetření vstupů a výstupů aplikace. Tento základní kámen bývá často také kamenem úrazu menších projektů, které nespoléhají na nějaký již naprogramovaný a prověřený framework. Tyto aplikace typu framework mívají většinou kvalitní architekturu (např. *MVC*), která díky týmu vývojářů není tolik náchylná na nejčastější bezpečnostní chyby a hlavně je při větší oblibě dostupná podpora: velice důležitá dokumentace a neméně důležité opravy příp. chyb. Příkladem může být např. framework od společnosti Zend[10].

Framework je ustálený výraz pro strukturu, která slouží jako podpora při programování, vývoji a organizaci jiných softwarových projektů [9]. Jeho výhody se projeví na velkých projektech nebo při opakovaném použití.

Základem ošetření vstupů se stává jednotné rozhraní pro filtrování. Naprogramoval jsem jednoduchou třídu, která se drží základních doporučení pro práci se vstupy. Tato třída *secureRequest* je v příloze, v adresáři *./Proof Of Concept/secureRequest* je navíc i formulář, který jsem používal při testování třídy.

Ze superglobálních polí extrahuje proměnné podle daného nastavení. Toto nastavení je dvojrozměrné pole, kde určujeme z jaké metody má proměnná přijít, s jakým názvem, datovým typem, dále určuje příp. defaultní hodnotu. Defaultní hodnota se inicializuje, pokud je poslední proměnná nastavena na true. Takto by vypadal příklad užití na proměnnou z *GET* metody (první řádek pole). Veškerý popis je okomentován v kódu v příloze.

```
1     <?php
2     $allowedVars = array(
3                         array ('GET','gId','I',0,true),
4                         array ('POST','pHeader','S','',false),
5                         array ('POST','pContent','S','',false),
6                         array ('FILES','fFile','F','',false),
7                         array ('POST','pSent','B','',false),
8                         array ('GET','gDouble','D',0,true)
9                     );
10
```

10 http://framework.zend.com/

```
11    $secureVar = new secureRequest($allowedVars);
12
13    //vypiseme promennou, pokud neprisla v GET datech vypise se zde defaultni hodnota 0
14    echo $secureVar->gId;
15    ?>
```

Tato třída uloží do svého privátního pole pouze ty proměnné ze superglobálních polí, které odpovídají dané matici resp. každém řádku z dvojrozměrného pole. Třída obsahuje metody pro

- ✔ nastavení hodnoty *__set()*,
- ✔ získání hodnoty *__get()*,
- ✔ zjištění, zda je proměnná nastavená *__isset()*,
- ✔ metodu pro zahození proměnné *__unset(),*
- ✔ kontrolu datového typu *validate()*.

Konstruktor třídy přebere dvojrozměrné pole *$allowedVars* a postupně ze superglobálních polí (*$_GET, $_POST, $COOKIE, $_FILES*) očistí a uloží do privátního pole pouze ty proměnné, které vyhoví danému nastavení v *$allowedVars*. Tato třída také testuje přes funkci *is_uploaded_file()*, zda-li byl soubor poslán a nebyl nijak skriptu podstrčen. Je třeba však upozornit, že tato třída se nestará o formát výstupních dat. Podle toho kam vypisujeme je třeba volat funkci *htmlspecialchars()* nebo projít regulárními výrazy či jinými technikami filtrování (více v části 5.2 XSS).

Kód je lehce znovupoužitelný pro jednotlivé skripty, ale pokud má být kontrola konzistentní, je třeba toto provést na každé stránce. Proto je tato třída více vhodná pro implementaci v nějaké framework aplikaci.

Existují navíc jiné možnosti jak se vypořádat se vstupy. PHP nabízí rozšíření filter (aktualizováno od verze 5.2), který slouží pro jednotnou kontrolu a úpravu dat, především pro kontrolu dat zadaných uživatelem a pro použití v situacích, kdy mají určité znaky speciální význam (např. v HTML nebo SQL) [10]. Organizace OWASP také nabízí třídu pro práci se vstupy a výstupy. Jedná se o projekt OWASP PHP Filters [11]. V další části si uvedeme příklady a řešení zabezpečení útoků, které využívají nezvalidované vstupy či výstupy.

5.2 Cross-site scripting (XSS)

Filtrování přijímaných dat již bylo dostatečně připomenuto a při tomto útoku je potřeba věnovat největší pozornost právě filtrování dat. Je třeba projít veškeré vstupy a povolit zápis pouze validních dat. Dále veškeré výstupy se musí projít, zda-li neobsahují HTML tagy nebo skripty, které by mohly narušit *layout* stránek nebo využít ke vložení škodlivého kódu. Tento kód by mohl např. sloužit k nepovolenému přístupu do administrační části obchodu.

Je velice složité (ne-li nemožné) filtrovat to, co je v datech nežádoucí. Technik [12] jak obejít různé regulární výrazy je mnoho a většinou využívá i bezpečnostních děr prohlížečů. V tomto ohledu lze uživatelům internetu pouze doporučit alternativní prohlížeče (Firefox[11] nebo Opera[12]) nebo minimálně používat nejnovější verze Internet Exploreru a stahovat nejnovější updaty. Chris Shiflett [13] varuje před používáním vlastních filtrovacích funkcí. Je velice pravděpodobné, že se opomene nějaký nepatřičný znak a tím se nezahrnou veškeré známé techniky útoku. V tomto je lépe použít podporovaných knihoven, buď přímo v PHP – rozšíření filter a funkce k tomu určené (*htlmspecialchars*(), *strip_tags()*

```
1   <?php echo htmlspecialchars($foo, ENT_QUOTES, "utf-8"); ?>
```

nebo např. již zmíněnou knihovnu OWASP PHP Filter.

Jsou ale případy kdy tato funkce nefunguje [14]:

```
1   <?php echo '<img src="'. htmlspecialchars($foo, ENT_QUOTES, "utf-8") .'" />'; ?>
```

Následný kód se provede, i když jsme použili funkci *htlmspecialchars():*

```
1   <?php
2    $foo = "javascript:alert(document.cookie);";
3    echo '<img src="'. htmlspecialchars($foo, ENT_QUOTES, "utf-8") .'" />';
4    ?>
5    výsledek bude <img src="javascript:alert(document.cookie)" />
```

Lepší než odstraňovat špatné znaky (používat černou listinu), je používat kritéria pro validní data resp. validní znaky. Např. v České Republice je PSČ pět čísel. Pomocí regulárního výrazu přesně otestujeme, že jsme obdrželi pět čísel. K tomuto slouží funkce *ereg()* resp. *eregi()*:

```
1   <?php if (eregi("^[0-9]{5}$", ($_POST["fak_psc"])) $formOk=true; ?>
```

11 http://www.mozilla-europe.org/cs/products/firefox/
12 http://www.opera.com/

5.3 Cross-Site Request Forgeries (CSRF)

Tento útok se často kombinuje s XSS útokem, ačkoliv samotný může způsobit dosti potíží. Potíž je v tom, že nevědomky vede útok sám uživatel, který se např. přihlásil do administrace. Je možné, že při surfování se mu načte průhledný obrázek, který má minimální velikost a jako zdroj uvede adresu:

```
<img src="http://admin.domena.tld/admin.php?akce=delZbo&id=1" />
```

Přihlášený administrátor tak sám pošle request, který by v tomto příkladě mohl např. smazat zboží s id 1. Útočník tak jednoduše obejde autentizaci, šifrované spojení nám teď také vůbec nepomůže. Není možné zjistit zda je tento požadavek legitimní. Řešení komplexní obrany proti tomuto útoku nejsou příliš jednoduchá, totiž ani *POST* data nejsou chráněna. Jednoduše javascriptem totiž pošleme požadavek např. v tagu <body>:

```
<body onload="document.form.submit();return false;">
```

Dobrým pravidlem je, že veškeré důležité operace by se měly provádět výhradně přes metodu *POST*. To jak jsme si ukázali bohužel samotné nestačí. Pokud chceme povolit jen určité skupině uživatelů, je bezpodmínečně nutná autentizace, ta nám však také nezajistí ochranu.

Při řešení vycházíme z kompromisu: budeme chránit pouze ty části kódu, kdy se zapisuje do databáze nebo maže – tedy veškeré operace metodou *POST* v administraci elektronického obchodu. Dále je dobré ochránit zákazníka před tím, aby někdo objednával zboží za něj. Ochranu tedy přidáme i do potvrzování objednávky.

S řešením přišel Chris Shiflett [13], který přidal unikátní token do každého požadavku posleného z formuláře. Při načtení stránky se pro přihlášeného uživatele vygeneruje unikátní token, který se posílá formulářem a také v *$SESSION* proměnné. Při přijetí tohoto požadavku stejným skriptem, kterým posíláme data, si uložíme starý token pro další kontrolu a vygenerujeme si nový, který se vloží do skrytého pole.

Po indikace poslání formuláře skrytým polem, zkontrolujeme zda souhlasí token posílaný *POST* metodou ve formuláři s tokenem, který jsme si uložili do session (v tuto chvíli již starý token). Abychom případnému útočníkovi situaci ještě více ztížili, přidáme vypršení tokenu. Po určité době token vyprší a při kontrole, prováděné porovnáváním tokenů můžeme zkontrolovat, zda náhodou token nevypršel.

Zde je ukázka generování tokenu:

```php
<?php
session_start();
//ulozime si stary token ze SESSION a vygenerujeme novy
if ( isset($_SESSION["token"]))
    $tokenOld = $_SESSION["token"];
$tokenNew = createHash($heslo,"token");
$_SESSION['token'] = $tokenNew;
$_SESSION['token_timeout'] = time() + 900;?>
```

Funkce *createHash()* vytvoří hash uživatelského hesla a času – dá se tedy očekávat unikátní token.

Takto by mohl vypadat společný formulář pro jedno konkrétní id:

```php
 <form action="" method="post" id="form1"
 onsubmit="doPOST(this,'submit.php', 'smazat')">
 <div>
 <input type="text" name="text"/>
 <input type="hidden" value="" name="akce"/>
 <input type="hidden" value="1" name="id"/>
 <input type="hidden" value="1" name="odeslano"/>
 <input type="hidden" value="<?php echo $tokenNew ?>" name="token"/>
 <input type="submit" value="odeslat"/>
 </div>
 </form>
```

O poslání se stará javascriptová funkce *doPost()*

```javascript
<script type="text/javascript" >
    function doPOST(form,loc,val){
        form.akce.value = val;
        form.action = loc;
        form.submit();
        return false;
     }
</script>
```

Samotný formulář můžeme díky javascriptu potvrdit klasicky tlačítkem *submit* nebo lze využít různých událostí (onchange, onclick,...) a javascriptové funkce *doPOST()* a posílat tak skrytý formulář z různých míst na stránce. Pro různé akce – mazání, editace, změna stavu objednávky atp. využijeme tento jediný formulář, kterému určíme hodnotu skrytého pole (*<input name="akce"/>)* pomocí javascriptu a tím rozlišíme akce, které uživatel chce udělat.

Celý tento skript je v příloze s názvem *CSFR – implementace* a navíc implementuje metodu [15], jak předejít dvojímu poslání POST dat, když by uživatel dal tlačítko zpět a také jak vypsat hlášku o zdaru či nezdaru provedené akce.

5.4 SQL Injection

Nejznámějším a také hodně starým útokem je *SQL injection*. Znovu se jedná o vložení škodlivého kódu útočníkem. V tomto případě se využívá syntaxe jazyka SQL. Dotaz v SQL na ověření hesla a přihlašovacího jména může mít následující tvar:

```
SELECT * FROM Zakaznik WHERE login='$login' AND heslo='$heslo'
```

v proměnné *$heslo* pak může útočník poslat:

```
$heslo = " ' OR 1=1 --";
dotaz vybere veškeré údaje z tabulky Zakaznik, podmínka 1=1 vždy vrátí true
SELECT * FROM Zakaznik WHERE login='neco' AND heslo=' ' OR 1=1 --'
```

Mezi základní techniky patří odkomentování a použití jednoduchých uvozovek a např. logického výrazu 1=1. Existuje jich více, ale není třeba je hledat, jelikož na všechny existuje řešení a prozradím, že mezi nejlepší systémové řešení patří parametrizované dotazy (prepared statements) nebo také databázové procedury [16].

5.4.1 Řetězec jako parametr

Pokud do dotazu vstupuje řetězec, jak tomu je v příkladu výše, řešením v PHP je funkce *mysqli_real_escape_string()*. Tato funkce upraví proměnnou pro danou databázi (funkce je závislá na mysqli) tak, aby dotaz byl bezpečný. Provede se tzv. escape znaků.

Toto řešení je v celém projektu e-shopu, jelikož již dříve jsem přepsal veškeré dotazy na rozhraní *mysqli,* bylo snazší doplnit pouze správné testování datového typu, příp. použít funkce *mysqli_real_escape_string()*.

```
1     <?php
2         $password = "";
3         $login = "adminovic' /* ";
4         //$login="' or login is not null";
5         //$login="' OR '1'='1";
6
7         //escape nepatricneho kodu
8         $login= mysqli_real_escape_string($spojeni, $login);
9         $password = mysqli_real_escape_string($spojeni, $password);
10
11        $query = "SELECT *
12                  FROM zakaznik
13                  WHERE login = '$login' AND
14                  password = '$password'";
15    ?>
```

5.4.2 Číslo jako parametr

Existují však funkce, které testují datový typ a ty nám při testu čísla vystačí. Jak jsem si výše ukázali u řetězce to samozřejmě neplatí. Lze použít funkci *is_numeric(), is_integer()* je v tomto případě nevhodné, jelikož vše co přijde přes požadavek je datovým typem string. Funkce *is_numeric()* je na toto testování zcela vhodná.

Lze také využít funkce *sprintf(),*která formátuje řetězec dle daného formátu. V uvedeném příkladě by se první dotaz s *sprintf()* provedl a $param by se rovnalo 24. Funkce totiž vezme, pokud je na začátku, pouze číslo a zbytek zahodí. Druhý způsob by neprošel přes podmínku, *is_numeric()* vrátí false.

```
<?php
    /*
     * testujeme integer
     * idealne napr. $param="24"
     *
     */
    $param="24 or login is not null";

    $query = sprintf("SELECT id_zak, jmeno
                      FROM zakaznik
                      WHERE id_zak=%d",$param);

    if (is_numeric($param)) {
        $query = "SELECT COUNT(*)
                  FROM zakaznik
                  WHERE id_zak=$param";
    }
?>
```

5.4.3 Prepared statements

Databázový systém MySQL od verze 4.1 podporuje tzv. parametrizované dotazy. Tento způsob psaní aplikace dozajista vymýtí jednou pro vždy útoky *SQL injection*. Samotné dotazování se totiž skládá z několika kroků. Ty někdy přinášejí i časovou úsporu při dotazování. První krok je příprava dotazu, další je nastavení parametrů a určení jejich datových typů. Poté se teprve samotný dotaz vykoná. Samotný parametr už není databází interpretován, prostě ho v uvedeném příkladě vezme jako obyčejný řetězec. Dalším krokem je metoda *store_result(),* ta nám připravený dotaz uzavře a je možno zjistit výsledky. Tady je právě ta výhoda úspory času při dotazování. Před samotným zavoláním metody *store_result()*, je možné nastavit více parametrů a vícekrát provést dotaz, než je připravený dotaz uzavřen. Toto je výhodné např. u vkládání (INSERT) do databáze. V cyklu se dá této výhody krásně využít. Na server se v cyklu posílají pouze parametry a příkazy k vykonání. Po skončení cyklu se pouze zavolá metoda *store_result()*.

Následující příklad využívá rozšířené třídy *mysqli*, *mysqli_stmt* a také třídy *Exception*. Můžeme tak použít v parametrizovaných dotazech metodu *fetch_assoc()*, která je obdobou funkce *mysqli_fetch_assoc()* a v originální třídě *mysqli_stmt* neexistuje. Pokud dojde s chybě, spadne podle typu chyby na výjimku. Použité jsou následující třídy: *mysqliExtended* a *exceptions* [17] (na přiloženém cd v adresáři *./tridy a funkce/*)

```
<?php
    /*
     * prepared statements
     * rozsirene tridy mysli_stmt a mysqli
     * metoda fetch_assoc - naplni asociativni pole nazvi poli
     * vlastni exception handler
     *
     */
    try {
        $db = new mysqliExtended();

```

```
12          $outside ="admin --";
13          if ($stmt = $db->prepare("SELECT *
14                                    FROM azakaznik
15                                    WHERE login=?")) {
16              $stmt->bind_param("s", $outside);
17              $stmt->execute();
18              $stmt->store_result();
19              while ($row = $stmt->fetch_assoc()) {
20                  echo $row["prijmeni"]. "<br />";
21              }
22              $stmt->close();
23          }
24          $db->close();
25
26      }
27      catch (SQLException $e) {
28          echo "SQL Error : ".$e->getSQLError()." in <hr><PRE>".$e->getSQL()."</PRE><hr>";
29      }
30      catch (DBException $e) {
31          echo "Database Error: ". $e->GetMessage();
32      }
33      catch (Exception $e) {
34          exception_dump($e);
35      }
36  ?>
```

5.4.4 PDO

Pro ty, kdo připravují webovou aplikaci na různé databáze (*MySQL, MSSQL, Oracle, PosgreSQL*) existuje rozšíření PDO (PHP Data Objects). Toto rozšíření sjednocuje rozhraní pro různé databázové systémy. Odpadá tedy při přechodu na jinou databázi složité přepisování funkcí, které jsou specifické pro jeden databázový systém. U *MySQL* jsou to funkce začínající na *mysql* nebo *mysqli* (modernější rozhraní) a u některých by pouhé nahrazení mysql za mssql, při přechodu na databázový systém *MSSQL*, nestačilo. Tyto funkce by se museli přepisovat na jiné specifické funkce.

Výhoda je také to, že můžeme vyvinout databázový přístup pouze jednou a poté ho implementovat v různých databázových systémech. Nebo jednoduše dopředu nevíme zda se systém v čase nezmění za výkonnější.

Toto rozhraní není zas až tak vyspělé a odladěné jako obdobná jiná rozhraní a tak tu a tam trpí špatnými ovladači k databázovému systému. Už nyní je však možno najít stabilní ovladače a jejich kvalita bude časem pouze stoupat.

Zde je ukázka jednoduchého parametrizovaného dotazu:

```
1   <?php
2       /*
3        * PDO
4        */
5
6       $pdo = new PDO('mysql:host=localhost;
7                       dbname=nazevDB',
8                       'user',
9                       'pass');
10      $stmt = $pdo->prepare("SELECT *
11                             FROM zakaznik
12                             WHERE login = ?");
13      $stmt->bindValue(1, $outside, PDO::PARAM_STR);
14      $stmt->execute();
15      $result = $stmt->fetch(PDO::FETCH_ASSOC);
16      echo $result["id_zak"];
17  ?>
```

5.5 Správa relací (sessions)

Jelikož v rámci protokolu *HTTP* neexistuje možnost jak identifikovat uživatele mezi jednotlivými požadavky, které jsou posílány serveru, vývojáři přišli s implementací pomocí malých textových souborů *Cookies*.

Cookies jsou rozšíření protokolu *HTTP* protokolu, aby zajistily udržení stavu mezi jednotlivými transakcemi. Bohužel to však s sebou přináší problémy ohledně soukromí uživatelů. Často se stává, že cookies jsou zneužity, aby monitorovali pohyb uživatele mezi jednotlivými servery. Jde o tzv. *tracking cookies*.

Zpočátku se proto doporučovalo mít cookies vypnuté. V tomto případě by se informace o stavu automaticky posílali přes URL. Tento koncept však přináší daleko více bezpečnostních rizik [18], a proto postupem času výhody cookies převážily. Stále však existují rizika spojená s použitím cookies a odpovědnost za správnou implementaci nese programátor webových aplikací.

Nejčastější bezpečnostní rizika spojené se sessions se dají vyjádřit termíny jako session stealing (session hijacking) a session fixation. Vysvětlíme si dále tyto pojmy.

5.5.1 Session stealing

Možností jak sebrat session je vícero. Nejjednodušší situace nastává, pokud oběť posílá identifikátor session přes URL. Útočníkovi stačí hlavička *Referer*, která se v PHP jednoduše získá z globální proměnné

```
<?php echo $_SERVER['HTTP_REFERER']; ?>
```

Tato hlavička uchovává *URL* předchozí navštívené stránky. Bohužel veškeré hlavičky se dají jednoduše generovat i v PHP přes funkci *header ()*, a proto není žádné spolehnutí na data v požadavcích, které přicházejí od uživatele. Cookies počínaje, *POST* a *GET* daty konče. Veškeré navštívené URL se navíc ukládají do *cache* a to jak u uživatele, tak po cestě.

Způsobů jak získat *referer* hlavičku je spousty a všechny jsou velice jednoduché. Co tedy v tomto případě hrozí? Útočník, který získá id session má přístup i k datům, které jsou s tímto id spojené. A nejčastější použití sessions pro svůj komfort a pro svojí pružnost je autentizace uživatele. Zde je tedy již jasné proč *id session* nepředávat přes *URL*.

Jak uchránit id session? Jediným jistým řešením je šifrované spojení. Jestli tuto možnost nemáme, je žádoucí možný útok minimalizovat resp. to útočníkovi co nejvíce ztížit. Chris Shiflett doporučuje [18] kontrolovat mezi jednotlivými požadavky zda se nezměnila hlavička *user-agent*.

Tuto hlavičku posílá prohlížeč spolu s požadavkem na jednotlivé stránky a identifikuje typ a verzi prohlížeče, např. takto pro prohlížeč Firefox 2.0.0.6:

```
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; cs; rv:1.8.1.6) Gecko/20070725 Firefox/2.0.0.6
```

Můžeme předpokládat, že uživatel, který se právě přihlásil nezmění v průběhu session svůj prohlížeč. A ztížíme příp. útočníkovi možnosti útoky tím, že budeme kontrolovat zda se po přihlášení tato hlavička nezměnila. Implementace je naznačena v samostatné kapitole 6.5 Autentizace.

Další možnost, jak získat jednoduše session id existuje díky fukci glob(). Tato funkce i při nastavení serveru na *safe_mod* a *open_basedir*, vypisuje chybové hlášky, které prozradí název prvního souboru [19], takto lze získat session id. Safe_mod bohužel poskytuje pouze omezenou bezpečnost pro servery, které sdílejí více domén na jednom místě. Řešení je znovu pouze na programátorech či administrátorech serverů, jelikož vývojáři PHP jazyka prohlásili tento problém za *bogus,* tedy neuznali ho jako chybu PHP.

5.5.2 Vlastní session handler

Řešení problému s funkcí glob je použít vlastní systém správy sessions. PHP nabízí více možností, jak session ukládat a kam. Defaultní nastavení ukládá sessions do souborů na server do adresáře /tmp. Další možnosti ukládání je uživatelské (*user*), přes zabudovanou *SQLite* databázi nebo *MM*. *MM* je určeno pro velice výkonný způsob ukládání session dat přímo do sdílené paměti serveru, je však třeba mít nainstalovaný *mm shared-memory module*.

My si tu popíšeme právě tu možnost uživatelské správy sessions. Využijeme v PHP zabudované modernější rozhraní mysqli databáze MySQL. Nejdříve si vytvoříme tabulku následujícím skriptem:

```
1   --vytvoreni tabulky
2   CREATE TABLE  sessions (
3       SID char(32) collate utf8_czech_ci NOT NULL,
4        expiration int(11) NOT NULL,
5       value text collate utf8_czech_ci NOT NULL,
6     PRIMARY KEY  (`SID`)
7   ) DEFAULT CHARSET=utf8 COLLATE=utf8_czech_ci;
```

Informací jak si napsat vlastní session handler je poskrovnu. Obzvláště na stránkách php.net v manuálu najdete o ukládání session dat do databáze pouze zmínku. Popis [20] jednotlivých funkcí uveřejnila společnost Zend, která také stojí za vývojem PHP již od čtvrté verze. Ve skriptu, který se nachází v příloze, je okomentováno, co která funkce dělá.

Pro integrovaní do jiných projektů stačí načíst soubor *./tridy a fce/session_handler.php* násle-

dující způsobem:

```
<?
//instalujeme session handler
require("./session_handler.php");

//obsah skriptu

//pred koncem skriptu
session_write_close ();
?>
```

Po posledním využití session (pro pořádek raději na konci skriptu) je třeba zavolat funkci *session_write_close()*, která zajistí uložení dat do databáze a uzavření spojení. Pokud používáte defaultní nastavení PHP, kdy se session ukládá do souborů na serveru, není třeba tuto funkci volat. PHP se o to automaticky samo postará před ukončením skriptu. U vlastního handleru volat funkci musíme.

Se session můžeme nyní pracovat jak jsme zvyklí. Tedy začátek session, vygenerování nového session id, získání id session a přiřazování hodnot do superglobálního pole je stejné:

```
<?php
session_start();
session_regenerate_id();
$sid = session_id();
$SESSION['foo'] = 'string';
?>
```

5.5.3 Session fixation

Existují 3 základní druhy zjištění session id:

- ukradení session (nejčastěji přes cookie)
- uhodnutí id (při správném nastavení serveru nejméně pravděpodobné)
- session fixation (fixace session id)

Zbývá nám tedy popsat poslední způsob jak útočník může získat session id. V PHP můžeme posílat inicializovat/udržovat session několika způsoby. Nejběžnější je přes výše zmiňované *cookies*, nebo také metodou *GET:*

```
<a href="http://domena.tld/index.php?PHPSESSID=1234">zde je odkaz</a>
```

Pokud se útočníkovi podaří pomocí XSS útoku vložit script do webové aplikace nebo do odkazu, může získat přístup právě přihlášeného uživatele [21].

Pomocí javascriptu [22]:

```
http://domena.tld/<script>document.cookie="sessionid=1234;%20domain=.domena.tld"</script>
```

Nebo pomocí metatagů [22]:

```
http://domena.tld/<meta%20http-equiv=Set-Cookie%20content="sessionid=1234;%20domain=.domena.tld ">
```

Vše se dá jednoduše maskovat[23], [12], [25] překódováním hodnoty za „http://domena.tld/“ do uživatelovi nečitelné formy.

PHP funkcí *session_start()* nerozlišuje, zda již session existuje nebo jestli pokračujeme v již započaté session. Toho útočník využije, tak že serveru vnutí své id session přes výše zmíněné metody. Poté stačí nastrčit oběti odkaz se session id, které zná a počkat až se přihlásí. Pokud se povede útočníkovi vložit pomocí XSS kód do aplikace, nemusí uživatele nutit kliknout na odkaz. Počká si, až se mu načte XSS javascript kód a až se uživatel přihlásí, jednoduše znalostí session id se „přihlásí“ do aplikace.

Jak se efektivně bránit rizikům spojených se session fixation [24], [18]:

- ✔ začít session až po přihlášení (pokud je to možné)
- ✔ po každé změně privilegií nebo přístupových práv vygenerovat nové session id pomocí funkce *regenerate_session_id()* - typicky po přihlášení uživatele
- ✔ zakázat propagování session id v *URL* – v php.ini nastavit "*session.use_only_cookies*" na 1

5.6 SSL

Mnoho problémů řeší šifrované spojení. Bez šifrovaného spojení není možné zajistit důvěrnost, nepopiratelnost, integritu a vysoce bezpečnou autorizaci. Bohužel přináší i nevýhody: vysoké zatížení serverů, více přenesených dat a tím pomalejší odezvy aplikace. V kritických bankovních aplikací je šifrování po celou dobu, kdy je uživatel přihlášen, nutností.

U elektronických obchodů často přichází kompromis – šifruje se posílání přihlašovacích údajů a přechod z košíku k zaplacení (potvrzení objednávky). Bohužel to mate uživatele jelikož některé prohlížeče varují, že jste přešli ze zabezpečeného spojení na nezabezpečené.

Také je problém s uznáváním podepsaných certifikátů. Uznávané certifikáty jsou drahé a pro některé malé internetové obchody se ekonomicky nevyplatí. Pokud máme na doménu nastavený certifikát prohlížeči obecně neuznávané certifikační autority znovu vyskakuje tabulka, která nezkušenému uživatelovi na důvěře v obchod nepřidá. Ne každý je ochoten zjišťovat komu certifikát patří, nebo dokonce zda-li je daná certifikační autorita důvěryhodná.

Pokud je bezpodmínečně nutné zajistit nejvyšší zabezpečení je třeba implementovat funkci, která zjistí jestli jsme na zabezpečeném šifrovaném spojení. Pokud nejsme, pokusí se na něj přesměrovat:

```php
<?php
function openSSL() {
    // overeni, zda pouzivame HTTPS
    $SSLPort=443;
    if ($_SERVER["SERVER_PORT"] != $SSLPort) {
        if (!isset($_GET["ssl"])) {
            header("Location: https://". $_SERVER["HTTP_HOST"].
                   $_SERVER["SCRIPT_NAME"].
                    "?". $_SERVER["QUERY_STRING"]."&ssl=1");
        }
        else
            echo "Zabezpečené spojení nelze navázat";
    exit;
    }
}
?>
```

Pokud používáme *sessions* společně s *cookies* je třeba posílat *cookies* také zabezpečeným kanálem *SSL*. Pokud bychom tak neučinili, vystavili bychom se nebezpečí útoku *session stealing*. Tento kód nastaví session cookie na zabezpečené spojení:

```
v php.ini: session.cookie_secure = 1
```

nebo funkcí, která nastaví životnost cookie na 60 minut a posílána bude pouze zabezpečeně:

```php
<?php
    session_set_cookie_params(time()+3600, '/', '', true, false);
?>
```

5.7 Autentizace

Popíšu zde způsob, jakým jsem implementoval autentizaci a jak jsem ji vylepšil. V projektu e-shopu byl požadavek, aby bylo možno procházet stránky bez registrace. Také byl samozřejmý požadavek, že vstup do administrace bude pouze pro správce obchodu. V autentizačních funkcích *login()*, *kontrolaPrihl()*, *idZak()* a potažmo i *typZak()* jsem s tím musel počítat.

Po ověření otisku hesla s otiskem formulářem posílaného hesla dojde k vytvoření session a cookie s názvem buď autorizace nebo admin. Takto se oddělí jednotlivé session. Administrátor tedy může být zároveň přihlášený jak do administrace obchodu, tak do samotného obchodu a to pod různými session. Použije se funkce *session_regenerate_id*(), která vygeneruje nové id session. Tento postup chrání aplikaci před *session fixation* útokem.

Do tabulky autorizace

```
CREATE TABLE  autorizace (
  id_ses varchar(100) NOT NULL,
  id_zak smallint(5) unsigned NOT NULL,
  cas int(10) unsigned NOT NULL,
  sec_id char(40) NOT NULL,
  PRIMARY KEY  (id_ses)
)
```

se uloží id session, zákazníka, aktuální čas funkcí *time()* a *sec_id* – což je hash id uživatele a toho, co browser uživatele posílá v hlavičce: *User-Agent*. Tento token se nesmí po dobu session změnit.

```
<?php $token = sha1($id.$_SERVER['HTTP_USER_AGENT']); ?>
```

V každém skriptu je funkce *kontrolaPrihl()*, která se spustí, pokud existuje správná cookie.

V této funkci se kontroluje, zda je uživatel přihlášen, zda je kombinace id session a sec_id v tabulce autorizace a pokud je, zda se nezměnil sec_id token (id uživatele nebo hlavička *User-Agent*).

Dále se kontroluje, zda-li není hodnota ve sloupci *cas* (čas), příliš malá a jestli tedy session už nevypršela. Pokud je vše pořádku vloží se aktuální čas do sloupce *cas* a funkce vrátí true.V opačném případě vrátí false a následuje poté vymazání z databáze a zničení session i cookie u uživatele. Rozlišuje se zde, zda jde o vypršení či něco jiného pomocí globální proměnné *$timeout*. Uživatel je tedy informován, v případě dlouhé nečinnosti, že mu vypršela session.

Funkce *kontrolaPrihl()* je v příloze, fce *login()* je příliš dlouhá a tak je tam pouze zkrácená verze.

5.8 Ošetření chybových výstupů

V mém projektu s ošetřením chybových výstupů není počítáno, jelikož na produkčním serveru je implicitně zakázáno vypisování chyb. U nově vznikajících projektů by však neměl již nějaký systém správy výpisu chyb chybět. Vše z jediného důvodu, veškeré výpisy chyb, které jsou příliš popisné mohou velice usnadnit práci útočníkovi. Takové techniky se často používá u *SQL injection*.

Vývojáři si vytvoří vlastní správu chyb a vypisují si pro testovací účely veškeré možné údaje. Toto by však mělo být na produkčním serveru zakázáno, nebo ještě lépe, měly by se chyby protokolovat do souboru, který není z venku pro kohokoliv dostupný.

V rozšířené třídě (*./tridy a fukce/mysqliExtended.php*) jsem použil erorr reporting (*.tridy a funkce/exception.php*) [17], který je rozšířením PHP třídy Exception. PHP od verze 5 totiž nabízí slušnou podporu výjimek. Třídu Exception, si tak může každý přetížit a napsat si vlastní error reporting např. i pro databázové rozhraní, jak je tomu v případě třídy *mysqliExtended.php*.

5.9 Standardy psaní kódu podle projektu PEAR

Neméně důležitým aspektem programování bezpečných aplikací, a proto ho tu musím zmínit, je přehlednost kódu a jednotný přístup při jeho vytváření. Je dobré se domluvit na jednotném stylu psaní a ten důsledně dodržovat nebo přejmout standard, který ustanovuje projekt PEAR. Tento standard má díky velké podpoře PEAR knihoven velkou ambici stát se obecným standardem, jak psát webové PHP aplikace. Proto, kde jsem měnil kód, jsem se snažil držet tohoto pravidla.

Základní pravidla[13]:

- ✔ jednotný styl psaní proměnných a funkcí – buď *$posliMail* nebo *$posli_mail*
- ✔ je dobré přidávat před samotný název proměnné její datový typ: např. *$iInteger, $sString, $aArray, $fFloat, $bBoolean*
- ✔ názvy proměnných by měly vyjadřovat akci, stav atp., nelze zkracovat $vypisTabulku na ~~$vypisTab~~
- ✔ názvy proměnných by měli vyjadřovat očekávaný pozitivní stav – *$jePoslan* ne ~~*$neniPoslan,*~~ při testování proměnné pak není jasné co očekáváme, př. *if (!$neniPoslan)*
- ✔ jednotný styl psaní závorek (složených, normálních)
- ✔ komentovat stylem phpDocumentor[14]
- ✔ používat <?php ?> ne zkrácené ~~<? ?>~~
- ✔ nemíchat v kódu různé jazyky, pokud možno psát anglicky

13 http://pear.php.net/manual/en/standards.php
14 http://www.phpdoc.org/

Kapitola 6

Testování

Důležitým krokem pro bezproblémové začlenění nových funkcí byla konstrukce testovacího modelu. Tímto modelem se prováděl rozbor technické proveditelnosti. Po důkladném otestování kusu kódu se projevily technické problémy, které se ihned řešily přepracováním modelu, jindy menšími změnami v kódu (viz *Proof of Concept* v příloze).

Testování nového kódu probíhalo postupnou integrací. Jednotlivé funkce či celé moduly byly zahrnuty do kódu poté, co se tento dílčí segment kódu důkladně otestoval. Každý segment byl krokován pomocí debug nástrojů, které již výše byli zmíněny. Testoval jsem nejdříve formou bílé skříňky. Zajímal jsme se o hodnoty uvnitř funkcí v průběhu parsování skriptu.

Po naprogramování více dílčích segmentů se provádělo testovaní metodou black box. Tedy z pohledu uživatele programu, který do kódu nevidí. Nejčastěji se takto testovaly hodnoty superglobálních proměnných z GET requestů. Testovalo se chování aplikace, pokud byla pomocí url vložena proměnná např. id (*http://domena.tld/nahled_zbo.php.php?id=1*) s nesprávným datovým typem – řetězcové hodnoty, které mohou umožnit sql injection. V tomto případě by se nemělo nic zobrazit a neměl by zde být ani výstup chybové hlášky s podrobným výpisem databázové chyby.

Kapitola 7

Implementace

Jelikož celý projekt by na papíře zabíral mnoho místa a jednotlivé techniky obrany proti známým útokům na webové aplikace byly již výše zmíněny, výsledná implementace do starého kódu je přiložena na CD v adresáři *./e-shop/*. V příloze jsou také důležité skripty, které jsem v této práci popsal, některé jsou zkrácené, ty kratší jsou nezkrácené.

Seznam použitých zdrojů

[1] WELLING, Luke, THOMSON, Laura. *PHP a MySQL : rozvoj webových aplikací*. Jan Kuklínek. 3. aktualiz. vyd. Praha : SoftPress, c2005. 830 s. ISBN 80-86497-83-6.

[2] *About The Open Web Application Security Project. [O projektu Open Web Application Security Project]. 2. 8. 2007*. [www dokument] dostupný z: http://www.owasp.org/index.php/About_The_Open_Web_Application_Security_Project

[3] The Open Web Application Security Project. *Ten Most Critical Web Application Security Vulnerabilies: 2004 [Deset nejzávažnějších bezpečnostních rizik webových aplikací]*. [cit. 27. 1. 2005]. [dokument ve formátu PDF] dostupný z: http://kent.dl.sourceforge.net/sourceforge/owasp/OWASPTopTen2004.pdf „The OWASP Top Ten" nabízí významný dokument o bezpečnosti webových aplikací. Reprezentuje široký konsenzus mnoha odborníků na bezpečnost SW, které jsou nejčastější chyby ve webových aplikacích.

[4] *The Open Web Application Security :* OWASP [online]. [cit. 2007-08-02]. The Open Web Application Security Project PHP Project. Dostupný z : http://www.owasp.org/index.php/Category:OWASP_PHP_Project

[5] *PHP Security Consortium* [online]. [cit. 2007-08-02]. Dostupný z: http://phpsec.org

[6] *The Open Web Application Security :* OWASP [online]. [cit. 2007-08-01]. The Open Web Application WebScarab NG Project Project. Dostupný z : http://www.owasp.org/index.php/OWASP_WebScarab_NG_Project

[7] CM311K1. *Zdrojový kód javascript scanneru Jikto.* [cit. 2007-08-03]. Dostupný z: http://www.security-portal.cz/clanky/zdrojovy-kod-javascript-scanneru-jikto.html

[8] *The Open Web Application Security :* OWASP [online]. [cit. 2007-08-02]. The Open Web Application Security Project WebGoat Project. Dostupný z : http://www.owasp.org/index.php/Category:OWASP_WebGoat_Project

[9] *Framework*. 20. *8. 2007. [www dokument] dostupný z:* khttp://cs.wikipedia.org/wiki/Framework

[10] VRÁNA, Jakub. *Filter a další novinky v PHP 5.2.* [cit. 2007-08-02]. Dostupný z : http://php.vrana.cz/filter-a-dalsi-novinky-v-php-5-2.php

[11] ZUCHLINSKI, Gavin, PRATT Jamie <jpratt@norwich.edu> a Hokkaido. *OWASP PHP* Filters. [www dokument]. Last updated: 4. 1. 2007. Dostupný z: http://www.owasp.org/index.php/PHP_Filters#Project_Overview Soubor PHP funkcí, které mají za cíl filtrovat uživatelské vstupy.

[12] RSNAKE. *XSS (Cross Site Scripting) Cheat Sheet Esp: for filter evasion.* [cit. 2007-08-02]. Dostupný z: http://ha.ckers.org/xss.html

[13] SHIFLETT, Chris. Foiling Cross-Site Attacks. *[Obrana před útokem Cross-site]. 14. 10. 2003.* [www dokument] dostupný z: http://shiflett.org/articles/foiling-cross-site-attacks

[14] The Open Web Application Security Project. *A Guide to Building Secure Web Applications and Web Services: 2.0 Black Hat Edition [Průvodce vyvíjením bezpečných webových aplikací a webových služeb: 2.0 Black Hat*

Edition]. [cit. 27. 7. 2005].
[dokument ve formátu PDF] dostupný z:
http://heanet.dl.sourceforge.net/sourceforge/owasp/OWASPGuide2.0.1.pdf
Tento návod je určen všem návrhářům, vývojářům, konzultantům a auditorům, kteří se zajímají o návrh, vývoj a implementaci bezpečných webových aplikací.

[15] VRÁNA, Jakub. *HTTP metody GET a POST.* [cit. 2007-08-02]. Dostupný z : http://php.vrana.cz/http-metody-get-a-post.php

[16] PHP. *SQL Injection*. 23.8. 2007. Dostupný z : http://php.net/manual/en/security.database.sql-injection.php

[17] K.LONDENBERG@LIBRICS.DE. 10. 2. 2005. Dostupný z : http://www.php.net/manual/en/ref.mysqli.php#49862

[18] SHIFLETT, Chris. The Truth about Sessions. *[Pravda o Sessions]. 15. 12. 2003.* Dostupný z : http://shiflett.org/articles/the-truth-about-sessions

[19] BRODERSEN, Peter. *PHP Session security flaw - bypassing safe_mode and open_basedir*. 9. 12. 2005. Dostupný z www: http://stock.ter.dk/session.php

[20] HERREN, John. *Trick-Out Your Session Handler*. 10. 5. 2006. Dostupný z : http://devzone.zend.com/article/141-Trick-Out-Your-Session-Handler

[21] Web Application Security Consortium. *Session Fixation.* [cit. 2007-08-05]. Dostupný z :
http://www.webappsec.org/projects/threat/classes/session_fixation.shtml

[22] Web Application Security Consortium. [online]. Web Application Security Consortium: Threat Classification. *Session Fixation.* 2004. [cit. 2007-08-02]. [dokument ve formátu TXT] dostupný z:
http://www.webappsec.org/projects/threat/v1/WASC-TC-v1_0.txt

[23] Cgisecurity.com. [online]. *The Cross Site Scripting (XSS) FAQ*. 5. 2002. [cit. 2007-08-07]. Dostupný z :
http://www.cgisecurity.com/articles/xss-faq.shtml

[24] *KOLŠEK, Mitja. Session Fixation Vulnerability in Web-based Applications, Version 1.0 – revision.* 12. 2002. akt.: 2. 2007. [dokument ve formátu PDF] dostupný z:
http://www.acrossecurity.com/papers/session_fixation.pdf

Příloha

V příloze uveřejním důležité skripty, na které se odkazuji z jednotlivých kapitol. A které lze jednoduše implementovat do dalších projektů. Veškeré další skripty včetně těchto jsou také na přiloženém CD nosiči.

Skript pro vypsání barevného kódu

Pro formátovaný výpis kódu jsem použil následující jednoduchý skript. Kvůli očíslování řádků jsem použil výpis do tabulky, jelikož textový procesor (*OpenOffice.org Writer*) to automaticky převedl také na tabulku.

./tridy a fce/highlight_code.php

```
<!DOCTYPE html PUBLIC "-//W3C//DTD
HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Bezpečnost v PHP - bakalářská práce - Daniel Císař</title>
<style type="text/css">
.num {
    float: left;
    color: gray;
    text-align: right;
    margin-right: 6pt;
    padding-right: 6pt;
    border-right: 1px solid gray;
}
code {
    font-size: 10pt;
    font-family: 'Times New Roman';
}

</style>
</head>
<body>
<?php
function highlight_num($file) {
      echo '<table style="background:#EEEEEE;"><tr>'.
           '<td width="30" class="num"><code class="textCode">';
    for ($i = 1; $i <= count(file($file)); $i++) echo $i."<br />";
    echo '</code></td><td>';
     highlight_file($file);
    echo '</td></tr></table>';
}

highlight_num('highlight_code.php');
?>
</body>
</html>
```

Proof of Concept – anti CSRF token

Tato zkouška konceptu dopadla dobře. Bohužel jiná zkouška, která měla zajistit vložení anti CSRF tokenu také do GET requestů ztroskotala. Koncept pro bezpečnost byl vynikající, zabezpečení vůči CSRF a session stealing bylo kompletní, bohužel však v praxi nepoužitelný. Při testech jsem narazil na zásadní nedostatek: uživatel nemohl dát zpět, jelikož ho to odhlásilo, resp. vypršela mu session. Skripty můžete pro inspiraci najít na cd (./token-no-back-click/).

Zde úspěšný model:

./Proof Of Concept/CSRF - Proof Of Concept.php

```
<?php
session_start();

//ulozime si stary token ze SESSION a vygenerujeme novy
if ( isset($_SESSION["token"]))
    $tokenOld = $_SESSION["token"];
$tokenNew = createHash($heslo,"token");
$_SESSION['token'] = $tokenNew;
$_SESSION['token_timeout'] = time() + 900;

//posilame formular
if (isset($_POST["odeslano"])) {
    //inicializace
    $formOk = true;
    //test dat formulare
    /*
     * pokud neni neco v poradku promennou
     * formOk nastavime na false
     */
    //TOKEN ANTI CSRF
    if ($tokenOld != $_POST["token"]) {
        $formOk = false;
    }
    //formular ok a token souhlasi
    if ($formOk) {
        /*
         * zapis do db atp
         */
        $hlaska .= "Zapis ok";
        $_SESSION['hlaska'] = $hlaska;

        //preposleme
        $cesta = $_SERVER['REQUEST_URI'];
        if (isset ($_SERVER['HTTPS'])) {
            if ($_SERVER['HTTPS'] == "on")
            $protokol = "https";
        }else $protokol = "http";
        header("Location: $protokol://$_SERVER[SERVER_NAME]$cesta", true, 303);
        //ulozime session do db a zavreme spojeni s db
        session_write_close();
        exit;
    }
}
?>
<?php echo '<?xml version="1.0" encoding="UTF-8" ?>';
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>Insert title here</title>
<script type="text/javascript" >
    function subForm(form,loc){
        form.action=loc;
    }
</script>
</head>
<body>
<?php
     if (isset($_SESSION['hlaska'])) {
```

```
62              echo $_SESSION['hlaska'];
63             unset($_SESSION['hlaska']);
64         }
65    ?>
66
67    <form action="" method="post" id="form1"
68    onsubmit="subForm(this,'submit.php', 'smazat')">
69    <div>
70    <input type="text" name="text" />
71    <input type="hidden" value="1" name="id"/>
72    <input type="hidden" value="1" name="odeslano"/>
73    <input type="hidden" value="<?php echo $tokenNew ?>" name="token"/>
74    <input type="submit" value="odeslat"  />
75    </div>
76    </form>
77    </body>
78    </html>
```

CSFR - implementace

```php
<?php
session_start();
//ulozime si stary token ze SESSION a vygenerujeme novy
if ( isset($_SESSION["token"]))
    $tokenOld = $_SESSION["token"];
$tokenNew = createHash($heslo,"token");
$_SESSION['token'] = $tokenNew;
$_SESSION['token_timeout'] = time() + 900;

//inicializace
if (isset($_POST['akce'])) {
    $akce = $_POST['akce'];
}
else $akce = "";

// posilame upravene udaje
if (isset($_POST['odeslano']) && $akce!='nacist') {
    $hlaska = "";

    //TOKEN ANTI CSRF
    if ($tokenOld == $_POST["token"] || $_SESSION['tokenTimeout'] > time()) {

        //posilame mazeme, pridavame, mazeme
        switch ($akce) {

            //mazeme
            case 'smazat':
                if (is_numeric($_POST['id'])){
                    $id=$_POST['id'];
                    $query=mysqli_query($spojeni,
                    "DELETE from novinky WHERE id_nov=".$id);
                }
            break;

            //upravujeme
            case 'editovat':
                    //kontrola prichazejici promenne
                    //akce v db
            break;

            //dalsi akce
        }

        //pokud se podarilo
        if (mysqli_affected_rows($spojeni)==1){
            if ($akce=="smazat")
                $hlaska .= '<div class="notice">Novinka byla vymazána.</div>';
            else
                $hlaska .=  '<div class="notice">Data byla uložena.</div>';
        }

        $_SESSION['hlaska'] = $hlaska;
        //preposleme
        $cesta = $_SERVER['REQUEST_URI'];
        if (isset ($_SERVER['HTTPS'])) {
            if ($_SERVER['HTTPS'] == "on")
                $protokol = "https";
        }else $protokol = "http";
        header("Location: $protokol://$_SERVER[SERVER_NAME]$cesta", true, 303);
        //ulozime session do db a zavreme spojeni s db
        session_write_close();
        exit;
    }
}
?>
<?php echo '<?xml version="1.0" encoding="UTF-8" ?>';?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>administrace</title>
<script type="text/javascript" >
    function doPOST(form,loc,val){
        form.akce.value = val;
        form.action = loc;
        form.submit();
        return false;
    }
</script>
</head>
```

```
82    <body>
83    <?php
84         if (isset($_SESSION['hlaska'])) {
85             echo $_SESSION['hlaska'];
86            unset($_SESSION['hlaska']);
87         }
88    ?>
89    <form action="" method="post" id="form1"
90    onsubmit="doPOST(this,'submit.php', 'smazat')">
91    <div>
92    <input type="text" name="text"/>
93    <input type="hidden" value="" name="akce"/>
94    <input type="hidden" value="1" name="id"/>
95    <input type="hidden" value="1" name="odeslano"/>
96    <input type="hidden" value="<?php echo $tokenNew ?>" name="token"/>
97    <input type="submit" value="odeslat"/>
98    </div>
99    </form>
100   </body>
101   </html>
```

Vlastní databázový session handler

```php
<?
// nazev tabulky
$sessionTbl = "sessions";
//z php.ini ziskame max dobu zivotnosti session
$lifetime = get_cfg_var("session.gc_maxlifetime");
/**
 * otevreme spojeni s databazi
 *
 * @param string $session_path
 * @param string $session_name
 * @return boolean
 */
function mysql_session_open($session_path, $session_name) {
    $db_user = 'user';
    $db_pass = 'pass';
    $db_host = 'localhost';
    $db_name = 'dBName';
    global $sessionDbConn;
    //pripojime k db
    $sessionDbConn= mysqli_connect($db_host, $db_user, $db_pass, $db_name);

    if (!$sessionDbConn){
        echo "Nelze se připojit k databázi";
        //mysqli_error($sessionDbConn);
        return false;
    }
    else {
        mysqli_query($sessionDbConn,'SET character_set_results=UTF8');
        mysqli_query($sessionDbConn,'SET character_set_connection=UTF8');
        mysqli_query($sessionDbConn,'SET character_set_client=UTF8');
        return true;
    }
}
/**
 * uzavreme spojeni s db
 *
 * @return mysqli object
 */
function mysql_session_close() {
    global $sessionDbConn;

    return @mysqli_close($sessionDbConn);
}
/**
 * precte data z databaze
 *
 * @param string $id
 * @return string
 */
function mysql_session_select($id) {
    global $sessionDbConn;
    global $sessionTbl;

    $id = mysqli_real_escape_string($sessionDbConn,$id);
    $query = "SELECT value FROM $sessionTbl
    WHERE SID = '$id' AND
    expiration > ". time();

    if ($result = mysqli_query($sessionDbConn, $query)) {
        if (mysqli_num_rows($result)) {
            $record = mysqli_fetch_assoc($result);
            //musime vratit string at je tam cokoliv
            settype($record['value'], 'string');
            return $record['value'];
        }
    }
    else return '';

}

/**
 * zapise data do db
 *
 * pokud jiz session id existuje, prepiseme novymi daty
 *
 * @param string $id
 * @param unknown_type $value
 */
function mysql_session_write($id, $value) {

    global $sessionDbConn;
```

```
82          global $sessionTbl;
83          global $lifetime;
84
85          $expiration = time() + $lifetime;
86
87          $id = mysqli_real_escape_string($sessionDbConn, $id);
88          $value = mysqli_real_escape_string($sessionDbConn, $value);
89
90          $query = "INSERT INTO $sessionTbl
91          VALUES('$id', '$expiration', '$value')";
92
93          $result = mysqli_query($sessionDbConn, $query);
94
95          if (! $result) {
96
97          $query = "UPDATE $sessionTbl SET
98          expiration = '$expiration',
99          value = '$value' WHERE
100         SID = '$id' AND expiration >". time();
101
102         $result = mysqli_query($sessionDbConn, $query);
103         }
104
105     }
106
107     /**
108      * mazeme data spojena se session id
109      *
110      * @param string $sessionID
111      */
112     function mysql_session_destroy($sessionID) {
113
114         global $sessionDbConn;
115         global $sessionTbl;
116
117         $query = "DELETE FROM $sessionTbl
118         WHERE SID = '$sessionID'";
119         $result = mysqli_query($sessionDbConn, $query);
120
121     }
122
123     /**
124      * mazeme data session, ktere vyexpirovaly
125      *
126      * @param unknown_type $lifetime
127      * @return unknown
128      */
129     function mysql_session_garbage_collect($lifetime) {
130
131         global $sessionDbConn;
132         global $sessionTbl;
133         global $lifetime;
134
135         $exp = time() - $lifetime;
136         $query = "DELETE FROM $sessionTbl
137         WHERE expiration < $exp";
138         $result = mysqli_query($sessionDbConn, $query);
139         return @mysqli_affected_rows($sessionDbConn);
140
141
142     }
143     /* nainstalujeme vlastni session handler */
144     session_set_save_handler("mysql_session_open", "mysql_session_close",
145     "mysql_session_select", "mysql_session_write",
146     "mysql_session_destroy",
147     "mysql_session_garbage_collect");
148     ?>
```

Autentizace – přihlášení uživatel

kontrolaPrihl()

```php
<?php
/*
 * fce, ktera kontroluje prihlaseni uzivatele,pokud neni
 * prihlasen nebo vyprsela session vraci fce false
 * pokud je vse ok do tabulky se zapise aktualni cas
 * globalni promenna timeout znaci vyprseni session
 */
function kontrolaPrihl(){
    // promenna zobrazForm je globalni a je defaultne nastavena na true,
    //pokud je uzivatel prihlasen nastavime na false
    //timeout nastavim na true, pokud mi vyprsi session
    global $zobrazForm;
    global $sid;
    global $jePrihlasen;
    global $timeout;
    global $spojeni;
    $timeout = false;
    $zobrazForm = true;
    $jePrihlasen = false;

    $cas = time();
    // za kolik sekund session vyprsi (30 min)
    $ad = $cas - 1800;
    if (isset ($_SESSION["sec_id"])) {
        $sesSecId = $_SESSION["sec_id"];
    }
    else $sesSecId = "";
    //je v tabulce session a neni vyprsela?
    $test = mysqli_query($spojeni,
            "SELECT id_zak FROM autorizace
            WHERE id_ses = '$sid' AND
            sec_id='$sesSecId'");

    $vyprsela = mysqli_query($spojeni,
            "SELECT id_zak FROM autorizace
            WHERE id_ses = '$sid' AND
            sec_id='$sesSecId' AND
            cas <= $ad");
    $numTest = mysqli_num_rows($test);
    $numVyprsela = mysqli_num_rows($vyprsela);
    if ($numTest == 0 || $numVyprsela == 1) {
        if ($numVyprsela == 1)
            $timeout = true;
        return false;
    }

    //nalezli jsem session a neni vyprsela
    else{
        $vyslTest = mysqli_fetch_assoc($test);
        $sec_id = createHash($vyslTest["id_zak"],"sec_id");
        $qHeslo = mysqli_query($spojeni,
            "SELECT heslo FROM zakaznik
            WHERE id_zak=". $vyslTest["id_zak"] ."");

        //nasli jsem zakaznika a nezmenil se hash browseru a id_zak
        if ( (mysqli_num_rows($qHeslo)==1) &&
             ($sesSecId == $sec_id)) {

                $vlozAkt = mysqli_query($spojeni,
                        "UPDATE autorizace
                        SET cas = $cas
                        WHERE id_ses = '$sid'");

                if ($vlozAkt){
                    $zobrazForm = false;
                    $jePrihlasen=true;
                    return true;
                }
                else {
                    $timeout = true;
                    return false;
                }
        }

    }
}
?>
```

Zkrácená funkce *login()*

```php
<?php
// prihlasime se, pokud souhlasi zahashovane heslo s heslem v dtb,
// login, heslo - predame z formulareout,
// admin 1, pokud se prihlasujem jako administrator shopu

function login($login,$heslo,$out,$admin) {
    global $sid;
    global $timeout;
    global $zobrazHlasku;
    global $spojeni;

        $heslo = MD5(mysqli_real_escape_string ($spojeni,$heslo) );
        $login = (mysqli_real_escape_string ($spojeni,$login) );

        //jako amdmin?
        if ($admin==1) {
            $vysledek = mysqli_query($spojeni,
                    "SELECT id_zak, zmrazen, registrovan
                    FROM zakaznik
                    WHERE zmrazen=0 AND
                    login='$login' AND heslo='$heslo' AND admin=1");
        }
        else {
            $vysledek = mysqli_query($spojeni,
                    "SELECT id_zak, zmrazen, registrovan
                    FROM zakaznik
                    WHERE login='$login' AND heslo='$heslo'");
        }

        //pokud tam takovy radek neni ukoncime
        if (@mysqli_num_rows($vysledek) <> 1) {
            return false;
        }
        //spravny login a spravne heslo,
        // zapiseme session id, id_zak a aktualni cas do db
        else {
            //oddelime nazvy session pro administraci
            //a eshop samotny
            if ($admin==1) $sessionName = "admin";
            else $sessionName = "autorizace";

            $res = mysqli_fetch_assoc($vysledek);

            //hash hesla a id zakaznika
            $sec_id = createHash($res["id_zak"],"sec_id");
            $cas = time();

            if ($res["zmrazen"] == 1) {
                //pokud se neprihlasi do 30 dnu od registrace
                // - ucet je zmrazen
                /*
                 * je potreba se prihlasit do 30 dnu od registrace
                 * zde testujeme zda se prihlasuje do 30 dnu
                 * pokud vse v poradku, prihlasime
                 */

                //kdyz prihlasil se pozde nebo je jeho ucet
                // od administratora zablokovany tak neprihlasime
            }
            //ucet nebyl zmrazen
            else {
                /*
                 * zacneme/vygenerujeme "novou" session,
                 * pokud jsem ji jiz nezacali vlozenim zbozi do kosiku
                 * nebo zmenou meny
                 * session fixation prevence
                 */

                    if (!isset($_SESSION["kosik"])
                        && !isset($_SESSION["mena"])) {
                        zacniSession($sessionName);
                    }
                    //session fixation prevence
                    session_regenerate_id(true);
                    $sid = session_id();
                $qInsertAutorizace=mysqli_query($spojeni,
                        "INSERT INTO autorizace
                        VALUES ('$sid', ".$res["id_zak"].",
                        $cas, '$sec_id')");
                /*
                 * uspesne jsme vlozili zaznam do tabulky autorizace,
                 * muzeme se opravdu prihlasit
```

```
                     * $_SESSION["prihl"]=1 znaci, ze bychom meli byt
                     * prihlaseni, resp. kontroluj me prihlaseni
                     */
                    if ($qInsertAutorizace) {
                        $_SESSION["prihl"]=1;
                        $_SESSION["login"]=$login;
                        $_SESSION["sec_id"]=$sec_id;
                        return true;
                    }
                    else {
                        session_kill();
                        return false;
                    }
                }
            }
    }
    ?>
```

Pozn. v originálním kódu je ještě předávaný parametr $out. Ten pokud je 1 značí, že se uživatel chce odhlásit. Není to logicky zrovna nejšťastnější řešení – mohla by to být samostatná funkce, ale toto nemá žádný vliv na bezpečnost a tak jsem ponechal předchozí stav.

Class secureRequest

```php
<?php
/*
 * retreat all data from POST, GET superglobal array
 * and validates its data type.
 * inputs: array of avaible variables names, data type
 */
class secureRequest
{
    /*
     * set $reportWarnings true if you wanna get warnings
     * of unknown request
     * good for debugging purposes
     */
    private $reportWarnings = false;
    private $secure = array();

    public function __construct($allowedVars) {
        //check alowedVars if there is
        //given $_GET, $_POST, COOKIE, FILES variable
        foreach ($allowedVars as $rowNum => $rowVal) {

            //default is not to automatically initialize variable
            if ($rowVal[4]!==true) $initialize = false;
            else $initialize = true;
            //reset the found state
            $found = false;

            switch ($rowVal[0]){
            // $_GET
            case 'GET':
                $superArray = $_GET;
                $superArrayName = 'GET';
            break;
            // $_POST
            case 'POST':
                $superArray = $_POST;
                $superArrayName = 'POST';
            break;
            // $_COOKIE
            case 'COOKIE':
                $superArray = $_COOKIE;
                $superArrayName = 'COOKIE';
            break;
            // $_FILES
            case 'FILES':
                $superArray = $_FILES;
                $superArrayName = 'FILES';
            break;
            //other
            default:
                $superArray = array();
                echo "<div><i>secureRequest Error</i> -
                     Unknown method: $rowVal[0]</div>\n";
            }

            foreach($superArray AS $varName => $value) {
                    //name is equal
                    if ($rowVal["1"]===$varName) {
                        //datatype is equal
                        $rowValType = $rowVal["2"];
                        if ($this->validate($value,$rowValType)) {
                            $this->secure[$varName] = $value;
                            //we found the right request
                            $found = true;
                        }
                        //bad datatype - throw error
                        else
                            echo "<div><i>secureRequest Error</i> -
                             Bad datatype in request: ".
                               $rowVal[0] ."['". $rowVal[1] ."']</div>\n";
                    }
                    //there is no variable in $allowedVars
                    elseif ($this->$reportWarnings)
                        echo "<div><i>secureRequest Warning</i> -".
                             "Ignoring unknown request: ".
                               $rowVal[0] ."</div>\n";
            }

            //we want to initialize
            if ($initialize && !$found) {
                //initialize! - default value
```

```
82                  $name = $rowVal[1];
83                  $this->secure[$name] = $rowVal[3];
84              }
85              //error
86              elseif (!$found) echo "<div><i>secureRequest Error</i> -
87                      No such variable in request: ".
88                      $rowVal[0] ."['". $rowVal[1] ."']</div>\n";
89          }
90      }
91
92      public function __get($name) {
93          if (isset($this->secure[$name]))
94              return $this->secure[$name];
95      }
96
97      /*
98       * Here is a possible hole
99       * you could set another datatype
100      * so look after what you set
101      */
102      public function __set($name, $val) {
103          if (isset($this->secure[$name])) {
104              $this->secure[$name] = $val;
105          }
106      }
107
108      /*
109       * ideal for testing if $secure['foo'] variable isset
110       * example: if (isset($secureRequest->gId))
111       * note $secureRequest is instance of class secureRequest
112       */
113      public function __isset($name){
114          return isset($this->secure[$name]);
115      }
116
117      public function __unset($name){
118          unset($this->secure[$name]);
119      }
120
121      /*
122       * validates data from posts and get request
123       * because everything from this is string
124       * we have to cast to string or test with
125       * is_numeric function the numerical expressions
126       *
127       */
128      public function validate($var, $type) {
129          switch ($type) {
130              //integer
131              case 'I':
132                  return
133                    (is_numeric($var) ? intval($var) == $var : false);
134
135              break;
136              //double
137              case 'D':
138                  if ((string)(float)$var === (string)$var)
139                      return true;
140                  else
141                      return false;
142              break;
143              //boolean
144              case 'B':
145                  if (is_bool($var) || $var=="true" || $var=="false")
146                      return true;
147                  else
148                      return false;
149              break;
150              //string
151              case 'S':
152                  if (is_string($var))
153                      return true;
154                  else return false;
155              break;
156              //uploaded file
157              case 'F':
158                  if (is_uploaded_file($var['tmp_name']))
159                      return true;
160                  else return false;
161              break;
162              //error
163              default:
164                  echo "Not valid data type<br />\n";
165                  return false;
166              break;
167          }
168      }
```

```
}
     /*
      * $allowedVars - array of settings for class secureRequest
         *         (string $method,
         *          string $name,
         *          string $type,
         *          mixed $default,
         *          boolean $initiate)
      *
      * $method   - possible values are: GET, POST, COOKIE, FILES
      * $name     - desired name of a variable
                    good practise is give beginning letter with desired method e.g. g,p,c,f
      * $type     - possible values are: 'I' as integer,
      *             'D' as double/float,'B' as boolean, 'S' as string
      * $default  - in case of $initiate=true, this is default
      *             value for variable wich wasn't initiated
      * $initiate - if true variable will be initiated if it wasn't
      *             from form
      */

$allowedVars = array(
                    array ('GET','gId','I',0,true),
                    array ('POST','pHeader','S','',false),
                    array ('POST','pContent','S','',false),
                    array ('FILES','fFile','F','',false),
                    array ('POST','pSent','B','',false),
                    array ('GET','gDouble','D',0,true)
                );

$secureVar = new secureRequest($allowedVars);

//not initialized with request but initilized with default value '0'
echo $secureVar->gId;
echo "\n<br /><br />\n";

//use setter carefully
$secureVar->gId =
    "setter doesn't go through validation process<br />";

//if we not sure we can still validate datatype
//result: not valid, we set gID to string
if ($secureVar->validate($secureVar->gID,"I"))
    echo "valid<br />";
else
    echo "not valid<br />";
echo "\n<br />\n";

//if we not sure we can still validate
//result: valid, we set gID to string
if ($secureVar->validate($secureVar->gId,"S"))
    echo "valid<br />";
else
    echo "not valid<br />";
echo $secureVar->gId;
echo "\n<br />\n";


//testing form sending
if (isset($secureVar->pSent)) {
    echo 'do some code here'."<br />\n";
}
echo "\n<br /><br />\n";

//simply getting the validated value
echo $secureVar->pHeader;

//getting not validated (we don't send this) value
//result we get no value
echo $secureVar->pName;
?>
```