FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Zabezpečení webové aplikace
Student:	David Pokorný
Vedoucí:	Ing. Přemysl Vaculík
Studijní program:	Informatika
Studijní obor:	Bezpečnost a informační technologie
Katedra:	Katedra počítačových systémů
Platnost zadání:	Do konce letního semestru 2019/20

Pokyny pro vypracování

Zanalyzujte podle požadavků webovou aplikaci a vypracujte analytickou rešerši k jednotlivým zranitelnostem:
- přechod a vynucení https; vytvoření testovací verze webu,
- vytvoření rolí pro uživatele webu; přiřazení pravomocí pro jednotlivé role,
- zabezpečení vstupů; bezpečné zpracování/vypisování uživatelem zadaného vstupu; podvrhnutí vstupu či identity.

Každou zranitelnost v rešerši popište, analyzujte možné dopady a navrhněte postup odstranění.
Po analýze implementujte opatření do webové aplikace moveandfight.com.

Seznam odborné literatury

Dodá vedoucí práce.

prof. Ing. Pavel Tvrdík, CSc.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 23. prosince 2018

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Zabezpečení webové aplikace

David Pokorný

Katedra informační bezpečnosti
Vedoucí práce: Ing. Přemysl Vaculík

12. května 2019

Poděkování

Chtěl bych poděkovat Ing. Přemyslu Vaculíkovi, vedoucímu mé bakalářské práce, za věcné připomínky k práci, opravu chyb v textu a celkovou psychickou podporu.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. Dále prohlašuji, že jsem s Českým vysokým učením technickým v Praze uzavřel licenční smlouvu o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona. Tato skutečnost nemá vliv na ust. § 47b zákona č. 111/1998 Sb., o vysokých školách, ve znění pozdějších předpisů.

V Praze dne 12. května 2019 ……

Odkaz na tuto práci

Pokorný, David. *Zabezpečení webové aplikace*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2019. Dostupný také z WWW: ⟨`https://Bagnat@bitbucket.org/Bagnat/bakalarska-prace.git`⟩.

Abstrakt

Cílem práce je seznámit čtenáře se základy bezpečnosti v oblasti webových aplikací se zaměřením na systém WordPress. Zvolené řešení jsou přizpůsobeny požadavkům malé firmy. Text je určen pro začínající programátory.

Klíčová slova zabezpečení webu, systém WordPress, validace uživatelských vstupů, HTTPS, XSS, session hijacking, WordPress Plugins

Abstract

The aim of this work is to introduce the reader to the basics of web application security focused on WordPress. The chosen solution is adapted to the requirements of a small company. The text is intended for beginning programmers.

Keywords web security, WordPress, user input validation, HTTPS, XSS, session hijacking, WordPress Plugins

Obsah

Seznam výpisů kódu

Úvod

U projektů, které jsou přístupné veřejnosti, je potřeba dbát na zvýšenou ochranu proti příchozím útokům z internetu. Špatně zabezpečený web může poskytnout útočníkovi dostatečné množství informací pro způsobení škody. Důvodem k útoku může být sběr dat o uživatelích, vložení vlastní reklamy apod.

Pro menší projekty mohou být tyto škody kritické a mít tak za příčinu nucené ukončení, kvůli příliš vysoké finanční škodě nebo kvůli prostému poškození jména následovaného úpadkem počtu aktivních uživatelů.

Jelikož se podílím na projektu MoveAndFight, kde se zatím neřešila bezpečnost, rozhodl jsem se v rámci této práce tuto chybu napravit. Rád bych aktuální nezabezpečenou pre-alfa verzi změnil na bezpečnou alfa verzi projektu.

V tomto projektu je nejdříve potřeba zprovoznit šifrování komunikace. Dále pro další vývoj je nutno vytvořit testovací verzi, na které lze měnit zdrojové kódy stránky.

První přímé zabezpečení webové aplikace bude validace uživatelského vstupu. S tímto bodem je zároveň i řešen problém správného vypisování uživatelských dat.

Jelikož budeme pracovat se systémem, který již disponuje řadou funkcí, je potřeba správně nastavit, kdo tyto funkce smí používat. Dále se zmíníme o bezpečnosti prostřednictvím hotových řešení třetích stran.

Cíl práce

V této práci se zaměříme na základní zabezpečení menšího webu. S projektem MoveAndFight se seznámíme hned v první kapitole, kde provedeme analýzu subjektu, který chceme zabezpečit. Potřebujeme znát především použité technologie při vývoji a technické zázemí.

Ve druhé kapitole se zaměříme na principy bezpečnosti webu jako celku. Stěžejní částí bude šifrované spojení mezi klientem a serverem. Při vývoji musíme dbát taky na implementační postupy, které si zároveň i navrhneme.

Poté se teoreticky seznámíme s častými útoky na web. Popíšeme si předpoklady pro úspěšný útok, rizikovost a možné dopady. Řekneme si, jak webovou stránku zabezpečit před těmito útoky. Ukážeme si, že stěžejní pro bezpečnost, je bezpečné přijeti a výpis uživatelských dat.

V další kapitole se již zaměříme čistě na redakční systém WordPress. Popíšeme si, jak tento systém řeší oprávnění, jaké kroky podniknout pro zamezení útoků a který další software nám pomůže s obranou.

Výsledná práce bude „kuchařka“, která připraví teoretické základy čtenáři a následně poskytne cestičku, která povede k zabezpečení webového projektu postaveném na redakčním systému WordPress. Jelikož je náš projekt zcela nezabezpečen, budu všechny postupy zabezpečení rovněž implementovat do projektu MoveAndFight.

Kapitola 1

Seznámení se s projektem

Pro lepší vyhodnocení rizik a implementace vhodných bezpečnostních mechanizmů je potřeba znát projekt, se kterým pracujeme a platformu, na které se pohybujeme. Proto jsem nucen nejdříve popsat samotný projekt a technické informace.

Projekt MoveAndFight (zkráceně MAF) je webová aplikace dostupná přes internetový prohlížeč. Projekt lze rozdělit na několik částí:

První část je redakční, která uživatelům nabízí informace jak správně cvičit, jak zdravě jíst a další obecné informace ke zdraví. Samotné informace o cvičení nemotivují lidi dostatečně, aby pravidelně cvičili. Proto nabízíme i cvičení formou hry. Zde uživatelé každý den splňují úkoly, za které dostávají herní body a vylepšují si svojí postavu – herní část webu.

Dále pro potřeby vzájemné interakce lidí je zde zbudována třetí část, kde formou příspěvků mohou vystavovat obsah zaměřený na zdraví – sociální část webu. Právě posledně zmiňovaná část je nejrizikovější právě díky možnosti nahrávat uživatelem definovaný obsah. Uživatelé mohou nahrávat textové příspěvky, fotky či dokumenty (v aktuální verzi dokumenty nejsou podporovány).

O vývoj stránek se starají dvě osoby. Já, David Pokorný, zajišťuji veškerý vývoj funkcionalit (PHP, databáze, JavaScript) a Tomáš Sedláček, který se stará o grafickou stránku projektu (HTML, CSS) a spravuje obsah webu.

Z technického hlediska je projekt provozován na webhostingu společnosti WEDOS Internet, a.s. (dále jen Wedos) na službě NoLimit. Verze PHP je 7.1, databáze je MySQL, kde mimo WordPress tabulek jsou i další tabulky využívané především herní částí. Stránky jsou vytvořené formou šablony v redakčním systému WordPress.

Domény `moveandfight.com` a `moveandfight.cz`, na kterých jsou produkční verze projektu, jsou taktéž spravovány společností Wedos. Momentálně `.com` doména je přesměrovaná na `.cz` doménu. Spojení probíhá přes protokol HTTP.

Veškeré úpravy se provádějí pouze na produkční verzi a není zřízeno žádné verzování zdrojového kódu.

Projekt je v tuto dobu (leden 2019) na úrovni, že veškeré funkčnosti pro testovací provoz (alfa verze) jsou hotové, ale úplně chybí veškeré zabezpečení. Lze tedy provést učebnicové útoky, které by jistě skončily úspěšně. Cílem bude tyto zranitelnosti odstranit a dostat projekt do stavu, kdy by existovala stabilní bezpečná verze.

Kapitola 2

Základní zabezpečení webových projektů

V této kapitole si popíšeme zabezpečení webu na úrovni celého projektu. Nebudeme tedy řešit jednotlivé soubory či vstupy. Základem všech webových projektů, které jsou ve vývoji, je verzování (spolu s ním si ukážeme i rozdělení testovací a produkční verze). Dále je důležitý oddělený provoz obou verzí a testování změn před nasazením. Po verzování začneme se zabezpečováním.

2.1 Verzování

Jak bylo zmíněno, projekt je třeba verzovat a dostatečně rozlišit ostrou verzi od testovací (nahráním netestovaných funkcí do produkčního prostředí dáváme útočníkovy příležitost nás ohrozit). Pro naše potřeby použijeme systém Git, který je zcela vyhovující. Jelikož verzování není předmětem této bakalářské práce, uvedu pouze myšlenku s jakou budeme v Gitu rozdělovat verze.

Nejprve pár slov o Gitu. Git je verzovací systém založený na ukládání změn v projektu. Data, která chceme verzovat, musíme nahrát do repozitáře, to jsou zdrojové kódy naší WP šablony a našich pluginů. Stav projektu v jistém okamžiku se nazývá revize (commit).

Revize tvoří jednosměrně zřetězený spojový seznam, což tvoří historii projektu. Novější revize ukazuje na své přímé předchůdce – rodiče. Na revizi může ukazovat také větev (branch). Větev je pouhý ukazatel na revizi, který slouží především k možnosti identifikovat nejnovější revize, protože na ty neukazuje žádná novější revize.

Větev dále může sloužit (jak název napovídá) k rozdvojení verze projektu. Toto se často využívá, pokud vývoj provádí více lidí. Každý vývojář si vytvoří svojí větev z hlavní větve, na které provádí změny. Když změny dokončí jednoduše proloží (z anglického merge) změny do hlavní větve. [1]

Z celého projektu (tj. WP šablona a naše vlastní WP pluginy) vytvoříme jeden repozitář, kde hlavní větev MASTER zastane ukazatel na aktuální produkční verzi. Vše co je pod MASTERem tedy bude přímo na doméně `moveandfight.cz`. Druhá, testovací, verze bude mít neustále svoji větev. Jelikož vytváříme alfa verzi, nazveme naší větev jménem Alfa. Po otestování proložíme změny z Alfa větve do MASTER větve a nahrajeme změny na webhosting. Toto řešení je velice jednoduché a pro práci dvou lidí na projektu je dostačující.

2.2 Vytvoření testovací verze

Nyní nastává problém, kde budeme provozovat testovací verzi. Nejvhodnější testovací verze je na lokálním stroji u každého vývojáře. V tomto případě útočník „nemá“ šanci, jak se k tomuto systému dostat.

Další možností, kde mít testovací verzi, je, ji provozovat stejně jako ostrou verzi. Tedy přístupnou pod nějakou doménou. Já volím tuto možnost, protože chceme do testování zapojit i „ne-vývojáře“, kteří si nedokáží zprovoznit lokálně web a ani nemají přístup ke zdrojovým kódům.

Vytvoříme tedy subdoménu `test.moveandfight.cz`. Testovací verze bude zcela oddělena od produkční verze. Nebude mít tedy žádné soubory společné s produkční verzí a bude mít vlastní databázi.

Separovanost obou verzí je vysoce důležitá. Útočník prostřednictvím testovací verze nemůže napadnout produkční verzi a zároveň v testovací verzi nebudou přítomny žádné osobní údaje hráčů.

Na Wedosu lze subdoménu vytvořit pouze s pomocí souboru `.htaccess`[1], kde nastavíme přesměrování adresy `test.moveandfight.cz` do adresáře `www/subdom/test` – toto je standardní řešení doporučené provozovatelem hostingu. [2]

Se změnou adresy webu musíme zajistit funkčnost odkazů a přesměrování. Protože WP používá absolutní odkazování, musíme změnit adresu webu v databázi a následně přegenerovat všechny odkazy v databázi. Herní část webu využívá relativní odkazy, takže ty řešit nemusíme. [3]

Poznámka: Ve větším projektu by nebylo vhodné takto zamýšlet implementační postup. Vhodnější je, mít verzi pro vývoj (lokálně u každého vývojáře), dále verzi pro testování s vlastní databází, následně předprodukční verzi, která již data využívá z produkční databáze a produkční verzi pro lidi. Předprodukční verze již nevyužívá testovacích dat, které mohou obsahovat systematické chyby způsobené generováním (např. obsahuje pouze hráče, kteří

[1] .htaccess je soubor, který nám dává možnost měnit nastavení webového serveru na úrovni adresáře.

potvrdili e-mail a již splnili alespoň jeden úkol). Pro naše účely stačí, rozdělit projekt na vývojovou/testovací verzi a produkční.

2.3 Zabezpečení domény a spojení

Nyní se zaměříme na zabezpečení přípojení klienta k serveru. Pro komunikaci se používá protokol HTTP, kde komunikace mezi zúčastněnými není šifrována. Libovolný útočník sledující komunikaci může odposlouchávat zprávy a dokonce i zasáhnout do komunikace. Pokud chceme šifrovat HTTP, stačí využít protokol HTTPS, který právě navíc využívá Transport Layer Security – označeno jako TLS.

2.3.1 HTTPS

HyperText Transfer Protokol Secured je protokol využívající ke komunikaci HTTP spolu s TLS. Tento protokol šifruje data asymetricky – musíme tedy rozumět Public Key Infrastructure (dále jen PKI). PKI je zjednodušeně způsob, jakým si klient může ověřit, zda je protistrana důvěryhodná.

U webových stránek se používá PKI pomocí certifikační autority (dále jen CA). Projekt, který si přeje vystavit certifikát k doméně, musí zažádat u CA o vystavení a následně dokázat, že se skutečně jedná o autorizovaný požadavek. CA poté vystaví certifikát, který podepíše svým soukromým klíčem. Pokud si chce někdo ověřit pravost certifikátu, musí, mimo jiné, zkontrolovat podpis certifikátu pomocí veřejného klíče CA. Tento veřejný klíč je buď nainstalován v počítači s certifikátem dané CA nebo ho lze stáhnout z jeho stránek. Dále se kontroluje expirace certifikátu, omezení definované v certifikátu, nepřítomnost v revokačním listu apod. [4]

Certifikát obsahuje všechny důležité údaje pro ověření, že odpověď skutečně přišla od serveru. Obsahuje samozřejmě název domény (kontrola, zda certifikát patří doméně), sériové číslo (kontrola, zda certifikát nebyl zneplatněn), expiraci, technické parametry (jako je verze certifikátu, použité algoritmy, …), vydavatele, veřejný klíč certifikátu (lze pak ověřit, že příchozí zprávy opravdu podepsala soukromým klíčem protistrana) a nejdůležitější část je podpis CA. Ten nám poslouží k ověření, že jsme nedostali podvržený certifikát.

Podívejme se na příklad našeho certifikátu:

Výpis kódu 2.1: MAF certifikát

```
Certificate:
   Data:
      Version: 3 (0x2)
      Serial Number:
         04:fd:[...]:65:9c:94:63:15:3f:00:37:e3:24:d5
   Signature Algorithm: sha256WithRSAEncryption
      Issuer: C=US, O=Let's Encrypt,
         CN=Let's Encrypt Authority X3
      Validity
         Not Before: Jan 22 14:57:03 2019 GMT
         Not After : Apr 22 14:57:03 2019 GMT
      Subject: CN=moveandfight.cz
      Subject Public Key Info:
         Public Key Algorithm: rsaEncryption
            Public-Key: (4096 bit)
            Modulus: 00:[...]:bd:85:59:72:ea:19
            Exponent: 65537 (0x10001)
      X509v3 extensions:
         [...]
         X509v3 Subject Key Identifier:
            30:07:05:55:[...]:0A:43
         X509v3 Authority Key Identifier:
            keyid:A8:4A:[...]:EC:A1
         X509v3 Subject Alternative Name:
            DNS:moveandfight.cz,
            DNS:www.moveandfight.cz, [...]
         X509v3 Certificate Policies: [...]

   Signature Algorithm: sha256WithRSAEncryption
      32:13:[...]:eb:15:ec:ef:4e:bb:e1:dd:38:48:61:ef:59
```

2.3.2 Implementace HTTPS

K vytvoření certifikátu jsou potřeba zejména následující věci:

- Server i klient musí podporovat HTTPS.
- Server musí mít k dispozici certifikát dané domény.
- Komunikace musí směřovat na `https://` místo `http://`[2].

Nejtěžší část je získání důvěryhodného certifikátu a následná instalace na server. Podepsání certifikátu CA může být drahá záležitost. Nabízí se nám několik možností, jak certifikát získat.

[2]Toto jiné směřování požadavku změní protokol, pomocí kterého se komunikuje a zároveň se mění výchozí port z 80 na 443.

Self-signed certificate je certifikát, který doméně poskytne asymetrickou šifru, ale neobsahuje podpis CA a z tohoto důvodu nelze ověřit pravost. Samotný certifikát samozřejmě podpis obsahuje, ale jedná se o podpis samotného tvůrce, který není důvěryhodný. Prohlížeč Chrome takovou stránku označí jako nedůvěryhodnou a před zobrazením vám vypíše populární hlášku „Your connection is not private“ s chybou `NET::ERR_CERT_AUTHORITY_INVALID` a pro pokračování musíte kliknout, že víte, co děláte. Tato metoda tedy nepřichází v komerční sféře v úvahu.

Vystavení certifikátu třetí stranou (CA) je způsob, že si sami zažádáme CA, aby nám podepsala certifikát. Tato možnost je zpoplatněna (od několika tisíc korun českých za rok [5]) a je třeba dokázat, že doménu vlastníte.

Nespornou výhodou těchto certifikátů je, že certifikát obsahuje i název organizace apod. Dále lišta s URL v prohlížeči zezelená. Tato možnost je vhodná pro větší organizace. Certifikát lze využít i pro jiné účely, než je zabezpečení komunikace na svých webových stránkách.

Let's Encrypt certifikát je certifikát vydaný organizací Let's Encrypt. Let's Encrypt je CA, která je zdarma a veřejně dostupná. Tato služba poskytuje certifikáty právě pro účely zapnutí HTTPS [6].

Pro naše účely je vyhovující, protože nám umožní zapnout šifrování a prohlížeč nezobrazí varování jako u self-signed certifikátu. Další nespornou výhodou je, že náš webhosting umožňuje zapnout Let's Encrypt přímo z administrace. Nemusíme tedy nic instalovat. Certifikáty se vydávají na tři měsíce a automaticky se obnovují.

Poslední nutnost pro nasazení HTTPS je změna URL adresy projektu, ale to jsme již dělali při vytvoření testovací domény – stačí tedy celý postup zopakovat.

2.3.3 Vynucení HTTPS

Vynutit HTTPS je snadné pomocí přesměrování. Každou příchozí komunikaci jednoduše přesměrujeme z `http://domena.cz` na `https://domena.cz`. Veškerá komunikace, kde URL začíná právě na `https://` využívá HTTPS protokol a je tedy i šifrovaná. K přesměrování máme dvě možnosti.

Jednoduchý způsob je využít `.htaccess` soubor. Tento konfigurační soubor umožňuje změny v nastavení serveru uvnitř adresáře hostingu. Lze pomocí něho nastavit podmíněné přesměrování, zakázání či povolení přístupu, definovat chybové stránky atd. My zde využijeme právě tu první možnost.

Webhosting musí mít povoleno využití htaccess funkcí. Mnoho webhostingů nabízí možnost výběru, zda chceme `.htaccess` využít či ne. Náš web-

hosting tuto možnost má, my ji tedy využijeme. Někteří poskytovatelé webhostingu povolí pouze základní změny přímo přes administraci hostingu.

Naše přesměrování v .htaccess bude vypadat takto:

Výpis kódu 2.2: Přesměrování pomocí htaccess

```
<IfModule mod_rewrite.c>
   RewriteEngine on
   RewriteCond %{HTTPS} off
   RewriteRule
      ^(.*)$
      https://%{HTTP_HOST}%{REQUEST_URI}
      [L,R=301]
</IfModule>
```

Kód je velmi jednoduchý. Pomocí `IfModule` se ptáme, zda je k dispozici knihovna rewrite, která poskytuje nástroje pro změny URL. Následuje zapnutí samotného modulu. `RewriteCond %{HTTPS} off` je podmínka, která je splněná právě, když není použit HTTPS protokol. Pokud HTTPS nepoužíváme provede se přesměrování. Před popisem posledního příkazu se podívejme, z jakých částí se v tomto případě[3] skládá URL:

```
{REQUEST_PROTOCOL}://{HTTP_HOST}%{REQUEST_URI}
```

`REQUEST_PROTOCOL` je protokol požadavku. Právě tuto část URL chceme změnit z `http` na `https`. Proto ji nastavíme natvrdo na `https`. Pod `HTTP_HOST` se skrývá celá doména, kam se klient připojuje. Ponecháním proměnné bude přesměrování fungovat i pro domény s TLD `.com` a pro testovací subdoménu `test`. `REQUEST_URI` jsou data odesílaná pomocí metody GET.

`RewriteRule` je příkaz, který přesměruje stránku shodnou s prvním parametrem na stránku ve druhém parametru. Jako první parametr máme regulární výraz `^(.*)$`. Stříška značí začátek, dolar značí konec. Tečka je substituce pro libovolný znak a hvězdička iteruje předchozí znak. První parametr tedy splní libovolná adresa.

Jako třetí parametr zde máme `[L,R=301]`. To jsou flagy, které poskytnou bližší nastavení přesměrování. Flag L znamená last, tedy po tomto přesměrování se nebudou další příkazy provádět a přesměruje se okamžitě. R je mód přesměrování. 301 značí „moved permanently“ a 302 „found (moved temporarily)“. Tento mód se odešle přímo klientovi jako návratový kód požadavku. Jelikož se stránky natrvalo přemístí na URL s `https:\\` použijeme 301.

[3] Říkám „v tomto případě“, protože URL lze členit podrobněji — dále lze zahrnout např. port (zde počítáme s výchozím portem 80).

Druhá možnost, jak zajistit vynucení HTTPS je přesměrování přes příkazy PHP. Touto metodou jsme schopni logovat přesměrování, popř. počítat pokusy o navázání nezabezpečeného spojení a po určitém počtu pokusů vypsat chybovou hlášku.

Tato metoda ovšem dává příležitosti k chybě. Musí být zaručeno, že se skript spustí vždy. Např. kdyby skript na přesměrování byl umístěn pouze na hlavní stránce. Útočníkovi v tomto případě stačí, aby šel přímo na jinou stránku a přesměrování se neuskuteční. Dále se přesměrování pomocí PHP špatně implementuje, pokud využíváme cachování stránek webu. Musíme zajistit, aby se cache po změně odkazů vyprázdnila a staré odkazy zahodila.

Kapitola 3

Zabezpečení vstupů

Nejvíce kritickou částí celého projektu je fakt, že uživatelé mohou provádět změny obsahu na stránkách (vkládání příspěvků, korespondence, …). Útočník má tedy možnost posílat požadavky, které jsou přizpůsobeny pro vyvolání nežádoucích akcí. Důsledky mohou vést k vylepšení účtu, na které nemá nárok (získání herní bodů, aktivace VIP účtu bez zaplacení, …), napadení cizího účtu (vkládání negativních příspěvků jménem jiného hráče, získání citlivých údajů, …) nebo narušení chodu webu (odepření služby – Denial of service).

V této kapitole si popíšeme nejčastější útoky. U každého si vysvětlíme teoretický základ útoku, ukážeme si, co lze úspěšným útokem získat a jak se bránit. Pro začátek se koukneme na samotný základ bezpečnosti – validace vstupních dat.

3.1 Validace vstupních dat

Uživatel internetu posílá data skrze načtený HTML formulář. Jelikož tento formulář je na straně klienta, může odeslat libovolná data. Důkazem tohoto je fakt, že k odeslání dat vůbec nepotřebuje formulář, ale může vygenerovat např. cURL požadavek.

3.1.1 Riziko nevalidace dat

Útočník je schopen (bez validace dat) dostat server do situace, ve které se začne chovat nečekaně. Může dojít k neoprávněným výhodám uživatele až po pád serveru/scriptu.

Mějme následující situaci: Uživatelé jsou schopni na server nahrávat profilové obrázky. Útočník je schopen tedy nahrát obrázek u kterého v hlavičce

změní rozměry na 2000x0. Obecně na webu nechceme ukládat příliš velké obrázky a proto tento obrázek budeme chtít zmenšit. Jelikož chceme zachovat poměr stran, musíme si zjistit poměr. Chceme tedy vypočítat 2000/0. To je ovšem nedefinovaná operace, která povede k výjimce nebo k zastavení škálování.

Při neošetření výjimky typu „Fatal Error“ skončí požadavek s chybovým kódem 500 – Internal Server Error. Ovšem v tomto případě se zobrazí pouze PHP Warning, který na produkční verzi webu bude zcela potlačen. Výsledný poměr nám skončí jako hodnota `Inf`, se kterou škálování určitě nedopadne dle očekávání – zde již záleží na implementaci škálování.

V potaz připadají dvě varianty: Pokud škálování vyvolá Fatal Error, script se ukončí a je možné, že vznikly nekonzistence v databázi (např. že jsme již uložili nový odkaz na profilový obrázek, nedokončili jsme řádně registraci apod.).

V druhé variantě by škálování vůbec neproběhlo, ale vypsalo by se pouze varování (či jen false v návratové hodnotě). Potom bychom obrázek uložili v původní velikosti a to s chybou v hlavičce. Zde vzniká možnost zaplnění místa na serveru nebo propagování nevalidních dat k dalším uživatelům.

3.1.2 Validace dat

Vstupy můžeme kontrolovat na dvou úrovních – u klienta a na serveru.

Kontrola u klienta může být vytvořena pomocí JavaScriptu (dále JS) nebo pomocí parametrů přímo v HTML tagu vstupu. Toto slouží spíše jako zamezení špatně vyplněného formuláře. Útočník může bez problému přepisovat zdrojový HTML kód nebo načtené JS soubory. Tato kontrola tedy nezabrání útočníkovi odeslat nevalidní data. Proto kontrola pouze u klienta není vůbec dostačující.

Kontrola na serveru je tedy naší volbou. Každá data, která od klienta přijdou musí být zkontrolována. Tedy očekáváme-li numerickou hodnotu zkontrolujeme, zda jsme obdrželi opravdu číslo, pokud očekáváme více hodnot, tak zda jsme dostali opravdu všechny apod.

Implementovat pouze kontrolu na serveru je bezpečné, ale pokud se uživatel opravdu splete, je obtížnější mu dát možnost opravy. Kromě faktu, že uživatel musí stránku načíst znova, musíme také zajistit, aby se stránka vrátila do původního stavu a formulář se předvyplnil podle dřívějšího zadání.

Pokud ale implementujeme obě kontroly a i přes to nám přijdou nevalidní data, dá se předpokládat, že uživatel musel podstoupit určité kroky, které tyto kontrolu vyřadily. Proto v této situaci postačuje vypsat chybu (samozřejmě bez podrobných dat chyby) a nezabývat se pohodlím uživatele. Samozřejmě toto jde pouze pokud je jisté, že uživatel má zapnutý JS (popřípadě, pokud používáme parametry u HTML tagů, tak že je podporují všechny majoritní

prohlížeče). Správným logováním chyb jsme schopni sledovat nejčastěji špatně odeslaná data a podle toho přizpůsobit kontrolu u klienta a na serveru.

Zde jsou protiopatření spíše zdlouhavá. Všechna data je třeba kontrolovat na datový typ a na rozsah (u číselného hodnoty to například může být interval a u řetězce znaků regulární výraz). Samozřejmě zde zasahuje do značné míry business logika webu, tj. např. pokud e-shop nabízí slevy, pak musíme kontrolovat ještě, zda má uživatel opravdu nárok na vyplněné slevy a zda kombinace slev je možná. V praktické části si přímo ukážeme možné kontroly a jak k problému přistoupit systematicky.

3.2 Cross-site scripting (XSS)

Nyní předpokládejme, že jsme dostali validní data a potřebujeme s nimi pracovat. V mnoha případech potřebujeme uživatelský vstup vypsat ostatním uživatelům (příspěvky, komentáře či prostý nick uživatele).

Případ, kdy je vstup nahrazen za škodlivý kód (u kterého útočník očekává, že se spustí) se nazývá Cross-site scripting (dále jen XSS útok). XSS útok se dá kategorizovat do třech skupin. Projdeme si postupně všechny typy a uvedeme, jak lze daný útok provést a jak se chránit. [7]

3.2.1 Lokální / XSS Type 0

Celý tento typ útoku se provádí pouze lokálně a narušená data neopustí prohlížeč. Útočník tedy naruší nějaký lokální zdroj dat (URL adresa, HTML data), který potom samotná stránka spustí. Pro demonstraci mějme tento případ:

Smyšlená webová stránka `vitejte.com/index.html` slouží pouze k přivítání příchozích. Jedná se pouze o statickou stránku (server ani nemusí podporovat php) s následujícím obsahem:

Výpis kódu 3.1: Ukázka stránky zranitelné na lokální XSS

```
1  <!DOCTYPE HTML>
2  <html><body>
3      <h1>Vitejte
4        <script>
5          document.write(
6            decodeURIComponent(
7              document.location.href.substring(
8                document.location.href.indexOf("name=") + 5
9          )));
10       </script>
11     </h1>
12 </body></html>
```

Stránka očekává, že příchozí uživatel má v URL uložený parametr `name` se svým jménem. Jediné co se na stránce stane, je vypsání jména umístěného v URL. Útočník ovšem může oběti poslat odkaz, kde jméno nahradí škodlivým kódem. Například následující odkaz

```
vitejte.com/index.html?name=<script>alert("XSS")</script>
```

bude stránka považovat za zcela normální jméno, vypíše skript jako normální text a tento skript se následně automaticky spustí prohlížečem. Nyní jsme dokázali spustit náš kód na straně oběti. Je tedy vhodné si říct, proč je toto riskantní.

3.2.1.1 Důsledky lokálního XSS

Důsledky spuštění skriptu na straně klienta mohou být velice kritické. Začneme od nejméně závažného případu – Denial of Service motivace (dále jen DoS). Odepření přístupu touto metodou je velice jednoduché. Skript buď jednoduše celou stránku nahradí prázdným řetězcem nebo např. odstraní cookies (to má za následek ztrátu přihlášení). Jelikož se jedná pouze o lokální XSS je pravděpodobné, že skript se spustí pouze a jenom jednou. Při jakémkoliv přechodu na jinou stránka se URL změní a vše funguje.

Horší varianta je, že útočník skriptem může simulovat dotazy, které byste vy mohli posílat. Může tedy odeslat požadavky na změnu vašeho nastavení (emailu a poté hesla). Může načíst jiné skripty, které jsou cílené pro napadení uživatele a vážně ohrozit soukromí. Může vaším účtem vytvářet nechtěné příspěvky, otevírat stránky s reklamou, využívat váš výkon pro těžení kryptoměny apod.

Nejhorší varianta je odcizení cookies. V těchto datech se ukrývá session ID, což je informace pro server, pomocí kterého vás server rozpozná. V důsledku zcizení a vložení tohoto SID do prohlížeče útočníka, se serveru bude zdát, že komunikuje s vámi. Útočník tedy bude okamžitě přihlášen pod účtem oběti – viz kapitola 3.4.

3.2.1.2 Protiopatření proti lokálnímu XSS

U tohoto útoku stačí zkontrolovat, zda na webu jsou místa, kde lze tyto metody použít. Nejlepší je minimalizovat počet nedůvěryhodných zdrojů a všechna vypisovaná data zobrazit jako pouhý text a ne jako možný HTML kód.

V případě potřeby lze použít nástroje pro odhalení XSS zranitelností. Lze použít online nástroje, rozšíření pro prohlížeče nebo celé aplikace. Za zmíňku stojí `OWASP Zed Attack Proxy`, což je bezplatný nástroj pro odhalení bezpečnostních zranitelností pro webové aplikace.

Dále máme k dispozici XSS Auditora. XSS Auditor kontroluje, zda stránka neobsahuje XSS útok. Pokud odhalí útok, zablokuje celou stránku. Například, v našem útoku s `vitejte.com` Chrome vyhodil chybovou hlášku

`ERR_BLOCKED_BY_XSS_AUDITOR`

a bez možnosti „na vlastní odpovědnost pokračovat“ odmítl vykreslit stránku. Firefox na druhou stranu s tímto útokem neměl problém a okamžitě zobrazil a spustil škodlivý kód – tedy na tuto funkci se nelze spolehnout.

XSS Auditora lze zapnout/vypnout v hlavičce požadavku pomocí

`X-XSS-Protection: 1; mode=block`,

což je zároveň i výchozí hodnota pro Chrome.

3.2.2 Stálý / XSS Type 1

Tato zranitelnost je u webů, u kterých je možné ukládat a vypisovat data uživatele. Jsou to obzvláště funkce sociálních sítí a fór (zprávy, komentáře, návštěvní kniha apod.). Útočník nechá server uložit místo textu spustitelný kód. Server poté na požádání dané zprávy kód vypíše a pokud nebyl nijak ošetřen, tak se u uživatele spustí.

Jak název říká[4] , tento kód na serveru je uložený a proto hrozí, že postihne více lidí a opakovaně. Opět si ukažme na našem smyšleném webu, jak lze tento útok provést pomocí návštevní knihy. Chceme pouze uložit jméno a komentář návštevníků a posledních pár vzkazů vypsat na obrazovku. Na serveru tedy máme script:

Výpis kódu 3.2: Ukázka stránky zranitelné na stálý XSS

```
1  <?php
2  if(isset($_POST["visitor"])){
3      //bezpečné uložení jména a poznámky do dtb
4      //můžeme data zkontrolovat na datový typ
5      [...]
6  }
7
8  $visits = [...] //načtení vzkazů z dtb
9  foreach ($visits as $visit)
10     echo "{$visit["name"]}: {$visit["comment"]}<br>";
11 /*
12  * Možný výstup na stránce:
13  * David: Děkuji za přivítání
14  * Přemysl: Moc pěkné stránky.
15  */
16 ?>
```

[4]Anglický název je Stored XSS nebo Perzistent XSS

Je vidět, že stránka je jednoduchá a přímočará. Dokonce uložení můžeme zabezpečit proti query injection. Problém ale nastává na řádku 10 – výpis textu pomocí procedury `echo`. Pokud by někdo místo poznámky napsal

`<script>$.ajax({url:"like.php/id=5"});</script>,`

byl by schopen donutit všechny příchozí dát like jeho příspěvku. Prohlížeč obsah v `<script></script>` nevypíše, takže uživatel ani nezjistí, že byl napaden.

3.2.2.1 Ochrana proti stálému XSS

Ochrana spočívá v nahrazení speciálních znaků HTML za znaky, které v prohlížeči nevyvolají žádnou akci. K tomu nám poslouží funkce `htmlspecialchars()`. Příklad použití této ochranyýpu:

Výpis kódu 3.3: Náhrada speciálních znaků na HTML entity

```
1  <?php
2  $string = '<script>$.ajax({url:"like.php/id=5"});</script>';
3  $data = htmlspecialchars( $string );
4  echo $data;
5  ?>
6
7  /*
8   * Zdrojový kód stránky:
9   * &lt;script&gt;$.ajax({url:&quot;like.php/id=5&quot;});&lt;
10  * /script&gt;
11  *
12  * V prohlížeči se vypíše:
13  * <script>$.ajax({url:"like.php/id=5"});</script>
14  */
```

Teď je na nás, kam tuto funkci umístíme. Můžeme text upravit ihned po přijetí nebo až při výpisu. Doporučit mohu neodkládat konverzi. Vzniká riziko, že při rozšiřování webu se na tento fakt zapomene nebo nějaký cizí kód (např. plugin) tyto hodnoty vypíše bez ochrany.

Lze taky doporučit používat templatovací šablony pro výpis, které tyto ochrany aplikuje za vás. Například Blade Template. Pokud chceme vypsat text použijeme notaci `{{ $name }}`, kde se okamžitě a sama aplikuje funkce `htmlspecialchars`. Speciálně pro neaplikování `htmlspecialchars` musíme volat `{!! $name !!}`, což samo o sobě vyzývá k opatrnosti.

3.2.3 Zrcadlený / XSS Type 2

Zrcadlený XSS[5] je velice podobný způsob útoku jako lokální XSS, ale data se získávají z dat od serveru. Server tyto hodnoty nijak nemá u sebe uložený, jen je vrací – odráží či zrcadlí.

Samotným zdrojem kódu bývá postranní kanál (jakým je například e-mail). Opět k dodání kódu lze využít odkaz, ale zde nepotřebujeme, aby sama stránka brala data z URL a pomocí JS je vypisovala. Zde využijeme služeb stránky, např. vyhledávání. Vyhledávání často používá parametr `q` (jako query) pomocí metody GET (data jsou uložená přímo v URL). Dotaz potom vypadá `vitejte.com/?q=XSS`, kde web začne vyhledávat „XSS“ ve svém obsahu. Patrně v případě, že nic nenalezne vypíše hlášku jakou je např. „Je nám líto, ale pro dotaz ‚XSS‘ jsme nenašli žádný výsledek.“

Nyní nám server zrcadlil data, která jsme zadali do URL, což je právě to, co útočník potřebuje. Stejně jako v předchozích XSS jednoduše nahradíme řetězec k vyhledání za škodlivý kód. Toto URL poté můžeme rozeslat emailem. Jelikož u odkazů lze skrýt samotný odkaz a zobrazit jen text, máme vysokou šanci, že na odkaz příjemce klikne.

3.2.3.1 Ochrana proti zrcadlenému XSS

Ochrana je úplně stejná jako v předchozím případě. Nevypisovat bez úpravy data, která přijdou od uživatele. Dále je důležité si neříkat „Tenhle vstup vypisuji pouze jeho autorovi. Pokud vloží škodlivý kód ublíží jen sám sobě.“ Tento názor je chybný.

Jak jsme si ukázali u vyhledávání, ne každý vstup, co přijde, musí nutně zadat odesilatel. I pokud máme vstup, který určitě zadal uživatel, nikdy nevíme, kdo jiný si ho zobrazí (např. zveřejní login s heslem a vytvoří „past“ pro lidi, kteří by se přihlásili – platí hlavně pro stálý XSS).

3.3 HTTP response splitting

HTTP Response Splitting je metoda, kde se využije hlavičky HTTP protokolu.

HTTP protokol funguje v textové podobě. Veškerá komunikace mezi serverem a klientem je „korespondence“, kterou může vést i člověk se serverem (toto umožnuje např. příkaz `telnet HOST http`).

Na začátku HTTP požadavku je hlavička, kde jsou uvedené podstatné informace pro komunikaci. Jednotlivé záznamy v hlavičce jsou ukončené řádkovým zlomem. Konec hlavičky tvoří prázdný řádek. Hlavička může vypadat například takto:

[5] Originální název je Reflected XSS, překlad autora

Výpis kódu 3.4: Příklad HTTP hlavičky

```
HTTP/1.1 200 OK
Date: Mon, 25 Mar 2019 17:34:07 GMT
Server: Apache/2.4.23 (Win32) OpenSSL/1.0.2h PHP/7.0.9
X-Powered-By: PHP/7.0.9
Content-Type: text/html; charset=UTF-8
[empty line]
```

Pokud server do hlavičky vkládá data, která můžeme ovlivnit, jsme schopni ukončit hlavičku, doplnit chybějící data (pořadí jednotlivých záznamů si můžeme zjistit předem) a podvrhnout celou stránku.

Dokonalým kandidátem se může zdát např. parametr

`Set-Cookie: name=David`,

kde často můžeme ovlivnit hodnoty. Ovšem hodnoty procházejí konverzí přes funkci `urlencode()`, která zamění konec řádku za znak[6].

Lepším kandidátem je parametr pro přesměrování:

`Location: nova.adresa.cz`.

Není vůbec ojedinělé vidět odkazy typu `adresa.cz/?redirect=reklama.cz`. Pro napadení stránky tímto typem útoku je potřebný jeden předpoklad – stránky nesmí mít detekci vkládání konce řádku do hlavičky.

Pokud stránka využívá příkaz `header("Location: adresa.cz")`, pak v případě vložení řádkového zlomu PHP vyhodí varování „Warning: … new line detected …“ a hlavičku nevloží. Tento útok je tedy možný na servery (či zařízení), která hlavičku neskládají pomocí této (nebo podobných) funkcí. Tento předpoklad pro útok je zároveň i vhodná protiobrana.

3.4 Session hijacking

Session ID (zkráceně SID) je identifikátor spojení uživatele se serverem. Každý HTTP spojení je zcela oddělený od předchozích a z pohledu serveru jsou tato spojení nerozeznatelná. Sjednocení více spojení s jedním uživatelem se nazývá relace neboli session. Server si ovšem musí ukládat dodatečné informace, které si pamatuje napříč relací. K tomu má dvě možnosti.

První možnost je pomocí cookie dat. Cookie data se ukládají u klienta a při spojení se pokaždé odesílají serveru. K těmto informacím může přistupovat i lokálně spuštěný kód, nebo uživatel, který je může měnit.

[6]Nutno podotknout, že funkce setcookie automaticky zavolá urlencode, ale setrawcookie ne. Obě metody ovšem využijí funkce header() (viz. dále)

Další možností jsou session data, což je ekvivalent cookie, ale jsou uloženy na straně serveru. Klient k nim tedy nemá žádný přístup (nemůže je číst ani měnit). Ovšem vzniká problém jak při příchozím spojení určit, které session se mají přiřadit. Řešením je, že se každé session soubory označí identifikátorem a právě toto SID se uloží do cookie. Při spojení klient sdělí svoje SID a server mu přiřadí data. V těchto datech nalezneme informace jako jsou, zda jste přihlášeni, pod jakým účtem jste přihlášeni apod.

Zcizením SID a nahráním do svého cookie dokážete zmást server, který si bude myslet, že jste původní uživatel. To znamená, že najednou budete přihlášeni (pokud byl původní vlastník SID přihlášen) aniž byste zadali login či heslo (to udělal právě původní uživatel). Tento útok se nazývá session hijacking attack.

Podívejme se na možné provedení session hijacking [8].

3.4.1 Fyzické zkopírování SID

Na toto provedení se podívejme ze dvou pohledů. První, triviální, způsob je fyzický přístup k počítači. Získat SID je otázka pár sekund. V prohlížeči Chrome např. stačí zmáčknout `ctrl+shift+I` a do konzole napsat příkaz `console.log(document.cookie);`. Ofotíme výsledek, který si později opíšeme do svého prohlížeče a (pokud se uživatel neodhlásil) máme účet ukradený.

V druhém pohledu je třeba vědět, že u klienta se cookie ukládá do souboru na disku. Například Chrome v OS Windows 10 má cookie uložené v adresáři

```
C:\Users\<username>\AppData\Local\Google
    \Chrome\User Data\Default\Cookie
```

ve formátu SQLite. Pokud jsme schopni napadnout cizí počítač a ukrást tento soubor, pak ho dokážeme využít a provést session hijacking. Stejně tak SID není obsaženo pouze v těchto souborech, ale i v mezipaměti počítače atd.

Zde z pohledu webu toho tolik nelze dělat. Uživatel by měl počítač udržovat zabezpečený. My můžeme přidat další kontroly, jako je například kontrola IP adresy, kontrola prohlížeče apod. a hledat podezřelé chování. V případě, že detekujeme podezřelé změny, zahodíme SID na serveru (stejný efekt jako odhlášení uživatele). Tyto kontroly pomůžou i proti následujícím způsobům, proto je již nebudu uvádět.

3.4.2 Session sidejacking/sniffing

V tomto případě k úniku SID nedojde na straně klienta, ale na samotném spojení. Jak bylo řečeno, klient při každém spojení posílá svoje cookie (vč.

SID) a pokud tato komunikace probíhá nešifrovaně, je možné ji odposlechnout. Je to tedy útok založený na odposlechu a podle toho se lze i bránit. [8]

Veškeré spojení je třeba šifrovat, aby nebylo možné vyčíst ani přihlašovací údaje ani SID. Je taky potřeba si uvědomit, že SID posíláme opravdu při každém spojení (směrem na naší doménu). A to včetně načítání obrázků a CSS/JS souborů. Je tedy třeba každý dotaz na náš server zabezpečit. Další fakt je, že SID se nám může přiřadit již při prvním načtení webu a ne až po přihlášení. Je tedy potřeba šifrovat již první spojení anebo po začátku šifrování zahodit staré SID – funkce `session_regenerate_id()`.

3.4.3 Session fixation attack

Tento útok má trochu odlišný přístup něž ostatní. Útočník se místo ukradení SID snaží naopak podstrčit jeho. Máme zde několik technik podstrčení, které jsou různé v závislosti na zdroji SID pro server.

Nejjednodušší způsob je, pokud server získává SID přímo z URL. V takovém případě stačí, když útočník pošle oběti odkaz tohoto typu:

```
adresa.cz/?SESSION_ID=[SID].
```

Pokud se oběť potom přihlásí (a server nezmění SID) útočník může pouze obnovit stránku a pokračovat jako on.

Dalším způsobem jak předat své SID je přes metodu POST (klasický webový formulář). Útočník buď může podstrčit data a vložit skrytý vstup do formuláře (tag `<input type=hidden name=session_id value=[sid]>`) nebo může podvrhnout celý formulář.

Také může podstrčit celou stránku, která vypadá podobně nebo pošle formulář přímo emailem, ve kterém vyzve uživatele k přihlášení. V tomto případě si samozřejmě může přímo uložit login a heslo, ale výhodnější je nechat oběť přihlásit se přímo na oficiální stránce pod jeho SID. Tímto způsobem uživatel ani nezjistí, že byl napaden, ale útočník takto dokáže obejít i vícefaktorovou autentizaci.

Posledním způsob je využití cookie. Jelikož je toto klasická technika, jak sdělit svoje SID serveru, je téměř jisté, že tato metoda bude podporovaná. Pro úspěch této metody tedy musíme do cookie umístit svoji hodnotu. To lze pomocí XSS, kde JS opět využije třídu `document.cookie`.

Dalším způsobem může být využití meta tagu v HTML. Tag

```
<meta http-equiv="[data]">
```

může dle dokumentace předávat cookie data[7]. Každopádně moderní prohlížeče využití tohoto tagu omezily. Při použití

```
<meta http-equiv=Set-Cookie content="sessionid=abcd">
```

Chrome do konzole vypíše „*Blocked setting the ‘ sessionid=abcd ‘ cookie from a ‘<meta>‘ tag.*“ a tag zcela ignoruje. Google nabádá uživatele, aby použili možnosti přímo HTTP hlavičky nebo zmíněný `document.cookie`. [10]

3.4.4 Cross-site scripting

Jak název napovídá, k tomuto útoku potřebujeme použít XSS útok (viz. kapitola 3.2). Pomocí XSS dáme na stránku oběti kód, který nám sdělí jeho SID.

V ideálním případě vygenerujeme požadavek na náš server, přes který sdělíme SID a náš server ho uloží. K tomuto ale potřebujeme mít povolené „sdílené zdroje odjinud“ (v aj. Cross-origin resource sharing, dále jen CORS). CORS je mechanismus, který se stará o sdílení zdrojů napříč doménami. Zdroji zde myslíme kaskádové styly, JS soubory, obrázky apod. CORS nám dává do hlavičky parametry, které prohlížeč příjme a respektuje jejich nařízení.

Důležitý parametr je `Access-Control-Allow-Origin`, který prohlížeči sdělí, které zdroje pro stránku jsou přípustné. Hodnota `*` povoluje všechny zdroje a hodnota `adresa.cz` pouze tuto jedinou adresu, odkud smí stránka brát zdroje. V tomto případě, pokud by JS chtěl SID odeslat na server útočníka, prohlížeč by požadavek zablokoval, protože by to byl nepřípustný zdroj.

V případě, že útočníkovi získat SID brání `Access-Control-Allow-Origin`, může využít pouze požadavků na server v tomto parametru. V tomto případě se nabízí využít služeb samotných stránek, jak si poslat jeho SID. Na fóru to může být komentář k článku, na sociální stránce zas soukromá zpráva.

Základní obranou před XSS Session hijacking je využít CORS, tzn. do hlavičky odpovědi HTTP dát

```
Access-Control-Allow-Origin: adresa.cz.
```

[7]Sama dokumentace HTML označuje toto za nevyhovující řešení a radí použít HTTP hlavičku. [9]

Dále platí všechna předchozí opatření. Pro připomenutí si ukažme souhrn základních protiopatření pro Session hijecking:

1. Veškerou komunikaci šifrovat pomocí SSL/TLS. Tímto zabráníme odposlechnutí (resp. podvrhnutí) SID pomocí pasivního (resp. aktivního) odposlechu.
2. Vygenerování nového SID po úspěšném přihlášení. Toto znemožňuje podstrčení SID, protože po přihlášení přestane podstrčené SID platit. Pokud se SID podstrčí znovu po přihlášení, bude to mít za následek odhlášení uživatele (server zapomene, kdo uživatel je). Po dalším přihlášení se ovšem znovu SID změní. Pokud to webová aplikace umožňuje, pak SID měnit častěji (ideálně po každém načtení).
3. K manipulaci s HTTP hlavičkou a se session používat knihovní funkce. Vyhnete se tím chybám typu HTTP response splitting, nízká entropie SID (útočník by mohl SID uhádnout) apod.
4. Kontrolovat, zda se nezměnily dodatečné informace o uživateli, jako je IP adresa, prohlížeč nebo operační systém uživatele. V případě podezřelé aktivity smazat SID.
5. Při delší neaktivitě odhlásit uživatele automaticky. Toto nemá vliv na možný rozsah útoku, ale efektivně zkrátí útočníkovi čas, po který může zaútočit. Dobré je informovat uživatele, aby se sami odhlásil při odchodu.
6. Využít defense in depth – tj. termín pro kombinování vícero zabezpečení dohromady. Útočník musí prolomit všechna opatření.

3.5 SQL injection

Dalším velice populárním útokem přes nezabezpečené vstupy je SQL injection. Toto je metoda, kdy do dotazu na databázový server vsuneme vlastní dotaz. Pro představu ihned uvedeme jednoduchý příklad.

Mějme následující dotaz:

```
SELECT user_id
FROM users
WHERE login = "login" AND passwd = "heslo"
```

Jak je vidět, pokud heslo a login jsou správné, databáze nám vrátí ID uživatele. Pokud ale uživatel zadá místo loginu text `admin";`, pak se dotaz vyhodnotí jako

```
SELECT user_id
FROM users
WHERE login = "admin";" AND passwd = "heslo"
```

Databáze tento dotaz bude interpretovat jako dva dotazy oddělené středníkem. Druhý dotaz samozřejmě vyvolá chybu, ale první vrátí ID samotného administrátora. Web si tedy bude myslet, že heslo a login patří administrátorovi a přihlásí jej.

Útočník tímto stylem samozřejmě může podstrčit mnohem víc. Například může místo loginu zadat `"; DROP TABLE users;`, čímž smaže celou tabulku s uživateli.

V ukázce víme, jaká je struktura databáze. Tuto obtíž útočník může překonat dvěma způsoby.

Pokud útočí např. na web založený na WordPressu, může předpokládat, že prefix tabulek nebyl změněn. Tímto okamžitě zná rozložení všech tabulek. Pokud by byl prefix změněn, dá se předpokládat, že bude ve tvaru `/[a-z][a-z0-9][a-z0-9]?_/`[8], což nebude stát mnoho pokusů pro zjištění pravého prefixu.

Druhá možnost, jak se útočník dozví strukturu tabulek, je přes databázi `information_schema` (platí pro MySQL databáze), kde je uložené rozložení všech dalších databází. Tato informační databáze má vždy stejné rozložení. Tedy speciálně pro WordPress by příkaz vypadal takto:

```
SELECT DISTINCT SUBSTRING(
          `TABLE_NAME` FROM 1 FOR ( LENGTH(`TABLE_NAME`)-8 )
     )
FROM information_schema.TABLES
WHERE `TABLE_NAME` LIKE '%postmeta';
```

Tento dotaz vrátí prefixy všech WordPress webů, ze všech databází, které jsou pod přihlášeným účtem k databázovému serveru. Proto změna prefixu **není** dostačující ochrana před SQL injection. Tato metoda může být dokonce považována za security through obscurity (bežpečnost skrze utajení). [11]

3.5.1 Ochrana před SQL injection

Pravou ochranou před SQL injection útokem je nahrazení speciálních znaků v SQL. V praxi používané rozhraní pro dotazy mají parametrizovaný dotaz. Ten spočívá v zadání SQL dotazu a teprve poté zadání dat. WordPress ke komunikaci s databází využívá globální proměnnou `$wpdb` typu `class wpdb`. Výše zmíněný příklad by tedy vypadal takto:

[8] Velká a malá písmenka lze zaměnit, protože SQL není case sensitive. Dále lze předpokládat, že na začátku nebude mít číselné označení, ale na konci je možné pro rozlišení (např. wp1_)

Výpis kódu 3.5: Použití $wpdb

```
<?php
global $wpdb;
$wpdb->query(
    $wpdb->prepare(
        "   SELECT user_id
            FROM users
            WHERE login = %s AND passwd = %s",
        $login,
        $passwd
    )
);
```

Výstupem metody `prepare` pro `$login = "' OR 1=1;"` je SQL dotaz

```
SELECT 1
FROM users
WHERE login = '\' OR 1=1;' AND passwd = 'heslo',
```

kde můžeme vidět, že vložené znaky, které jsou speciální pro SQL jsou správně neutralizovány pomocí zpětného lomítka.

Tímto jsme probrali základní útoky na webové aplikace, ke kterým lze dojít nesprávnou validací vstupů a špatnou interpretací dat. Nyní se zaměříme přímo na systém WordPress, který je stěžejní pro projekt MoveAndFight.

Kapitola 4

Bezpečnost systému WordPress

V této kapitole se podíváme, na kterých základech je postavena bezpečnost WP.

4.1 Role-based access control

WordPress je postaven na role-based access control (dále jen RBAC). Každému registrovanému uživateli stránek je přiřazena právě jedna role, která je svázána s oprávněními (v aj. capability). Na základě přiřazené role se vyhodnocuje, zda je uživatel autorizován provádět dané úlohy.

Jelikož je WP redakční systém, tak má přednastavené základní role rozdělené podle správy článků. Od nejnižších oprávnění zde máme:

subscriber je schopen pouze číst články

contributor spravuje pouze vlastní články a nemůže je sám publikovat,

author ten od contributora může vydat vlastní články,

editor může spravovat i cizí články,

administrator správa WP webu,

super-admin správa multisite WP[9].

Pro představu základních oprávnění zde uvedu přehled nejčastěji používaných:

read smí vstoupit do administrace a měnit svůj profil

edit_posts upravovat články

publish_posts zveřejnit článek

[9]Multiple-site WP je jedna instalace WP, která zastřešuje více redakcí.

edit_page upravovat stránky

upload_files nahrát soubory (k článku či stránce)

moderate_comments cenzurovat komentáře

manage_option přístup do administrátorského menu

update_core/theme/plugin možnost aktualizovat WP/šablonu/plugin

install_plugins možnost nainstalovat plugin

WP je založen na systému článků a stránek. Máme zde výchozí typ článků pojmenované post. Dále můžeme definovat vlastní typy – tzv. custom post. Například MAF má takto definované mnoho typů článků - Cviky, Úkoly, Doplňky, Příspěvky apod.

Všechna oprávnění spojená s postem jsou svázaná přímo s build-in post. Proto musíme ke každému vlastnímu typu definovat vlastní oprávnění[10]. To snadno zařídíme pomocí parametru `capabilities` při definování typu článku.

Výpis kódu 4.1: Přiřazení názvu oprávnění k jednotlivým akcím článku

```
1  <?php
2  $arg = array(
3      [...],
4      'capabilities' => array(
5          'edit_post'             => 'edit_book',
6          'read_post'             => 'read_book',
7          'delete_post'           => 'delete_book',
8          'edit_posts'            => 'edit_books',
9          'edit_others_posts'     => 'edit_others_books',
10         'publish_posts'         => 'publish_books',
11         'read_private_posts'    => 'read_private_books',
12         'create_posts'          => 'edit_books',
13     )
14 );
15 register_post_type( 'book', $args );
16 ?>
```

Jelikož, předem vytvořené role ve WP jsou dost přizpůsobené k výchozímu typu článků, tak si nadefinujeme vlastní role. Využijeme pouze administrátora, který může vše. Nejprve odebereme role, které nepotřebujeme.

[10]Pokud tak neučiníme bude se ve výchozím nastavení používat pro custom post oprávnění jako pro build-in post.

Výpis kódu 4.2: Odebrání role

```php
<?php
//kontrola, zda role existuje
if( get_role('subscriber') ){
    remove_role( 'subscriber' );
}
?>
```

A následně vytvoříme role, které chceme používat.

Výpis kódu 4.3: Přidání role

```php
<?php
$result = add_role(
    'role_name',
    __( 'Role name' ),
    array(                          // výčet oprávnění
        'read'         => true, // povolit oprávnění
        'edit_posts'   => true,
        'delete_posts' => false // explicitně zakázat
    )
);
if ( null !== $result )
    echo 'Role created';
else
    echo 'Role already exists';
?>
```

Již během přidání role lze nastavit potřebná oprávnění. Lze ale i později přidat/odebrat oprávnění pomocí metody `add_cap( $role, $capability )`.

Výpis kódu 4.4: Přiřazení oprávnění

```php
<?php
// global class wp-includes/capabilities.php
global $wp_roles;
$wp_roles->add_cap( $role, $cap );
?>
```

Změny nastavení spojené s rolemi a oprávněními se ukládají do databáze, proto je stačí spustit pouze jednou.

Pro rychlejší a přehlednější řešení rolí je možné použít pluginy. Jelikož oprávnění jsou jen bool hodnoty, jsou tyto pluginy řešené způsobem zaškrtávacích boxů. Ze seznamu oprávnění stačí zaškrtnou požadované oprávnění k požadované roli a uložit.

4.2 Instalace WP

Nyní se podíváme, na které záležitosti si musíme dát pozor při instalaci a při počátečním nastavení systému WordPress.

4.2.1 Změna výchozích hodnot

Instalace WP probíhá ve třech krocích. Nejdřív se stáhne WP archiv, který se rozbalí a nahraje na server. Poté se provede konfigurace, kde nastavíme jazyk, název, přístupové údaje k databázi a vytvoříme účet s právy administrátora. Konečná fáze je instalace šablony a pluginů.

Při instalaci jsme vyzváni k zadání údajů o dtb, vč. předpony pro tabulky. Tuto předponu je vhodné změnit. Změnou této hodnoty způsobíme, že útočník již nebude znát názvy tabulek. Názvy tabulek si samozřejmě může zjistit z meta tabulek databázového serveru, ale je to určitá komplikace.

V další části si vytváříme účet, který bude sloužit jako admin. Je vhodné nepoužívat loginy jako jsou „admin“, „administrator“, „root“ apod. K prolomení účtu jsou zapotřebí dvě informace – jméno účtu a heslo. Není tedy zapotřebí vyzradit polovinu informací. Ze stejného důvodu je vhodné, aby články a stránky nepsal přímo administrátor (pokud se na webu vypisuje autor článku/stránky).

Je také vhodné vyzkoušet, zda adresa `domena.cz/?author=1` nám neprozradí login administrátora. Popřípadě můžeme vytvořit nového administrátora a poté degradovat uživatele s ID rovno 1.

4.2.2 Skrytí dodatečných informací

Každá informace, kterou poskytujeme veřejnosti může napomoci k prolomení naší bezpečnosti. Jednou z důležitých informací je verze WP, kterou používáme.

Zde bych rád podotknul důležitost mít aktuální verzi WP. WordPress.org vydává s novou verzí WP tzv. security log, ve kterém jsou popsány bezpečnostní záplaty. Tímto způsobem útočník dostane seznam zranitelností pro neaktuální WordPress stránky.

Aktuální nainstalovanou verzi se nejlépe útočník dozví z meta tagu v hlavičce HTML. Např. přímo ve zdrojovém kódu stránky `moveandfight.cz` můžeme nalézt tag

```
<meta name="generator" content="WordPress 5.1" />.
```

Tento tag odstraníme, když umístíme následující funkci do `function.php` v naší šabloně:

Výpis kódu 4.5: Odebrání generator tagu

```
<?php
function remove_generator_tag() {
    return "";
}
add_filter( "the_generator", "remove_generator_tag" );
?>
```

Dalším způsobem, jak zjistit verzi je ze souboru `readme.html` a `license.txt` v rootu projektu. Tyto soubory můžeme smazat nebo k nim odepřít přístup. Zde se sice přímo nenachází číslo verze, ale lze najít rozmezí verzí, kdy tyto soubory byly používány.

Velice důležité je deaktivovat vypisování souborů a složek. Pro přesvědčení, zda máme tuto funkci zapnutou stačí zadat `adresa.cz/wp-content/`. Pokud se nám ukáže seznam souboru a složek v tomto adresáři, musíme změnit nastavení serveru. Na hostingu Wedosu je toto ve výchozím nastavení zakázané, proto to nemusíme řešit (rozhodně neplatí, že každý hosting má stejnou výchozí hodnotu) [12]. Podobně by se mělo zakázat přistupovat ke skrytým souborům (začínající na tečku – např. `.htaccess`). Toto je ve výchozím nastavení zakázané přímo ve web serveru.

4.2.3 Deaktivace nebezpečných funkcí

Nejenom výpis dat nám může na bezpečnosti uškodit. Máme zde funkce, které můžou být pro funkčnost stránek kritické. Jednou z takových funkcí je editor šablony, který je přístupný z administrátorského menu. Dle mého by neměl být tento editor přístupný ani adminovi.

V tomto editoru mohu upravovat všechny soubory šablonám, vč. `.php` souborů. Pokud by útočník měl možnost tuto funkci využít, okamžitě končí veškerá bezpečnost – pomocí skriptů si může vypsat název, login a heslo k databázi, může mazat/měnit hesla, může se stát administrátorem a obecně již může cokoliv.

I když tuto funkci může použít jen administrátor, stále hrozí riziko, že by se rozhodl tímto editorem upravit `.php` soubor. Pokud by, byť jen omylem, udělal syntaktickou chybu a soubor uložil, při dalším načtení webu by vyskočila chyba 500 a znovu by nemohl soubor upravit. Vznikl by tak čas, kdy by byla webová služba nedostupná. Tento editor zakážeme přidáním řádku

```
Define("DISALLOW_FILE_EDIT", true);
```

do souboru `wp-config.php` v rootu projektu.

Další okno k informacím tvoří debugovací systém WP a serveru, na kterém jsou stránky provozovány. V produkční verzi projektu je tedy potřeba vypnout

zobrazování chyb pomocí `error_reporting(0);` pro PHP a pro WP příkaz `define( 'WP_DEBUG', false );`. Pokud necháme tyto funkce zapnuté, uživatelé budou v případě chyby informováni o kódu v místech chyby, popř. o SQL dotazu, který selhal. Toto by byly velice cenné informace pro útočníka. Jelikož debug mode bude patrně zapnutý na testovací verzi, je vhodné, aby útočník na ni neměl vůbec přístup. Může si totiž útoky (i s výpisem) vyzkoušet na testovací verzi a poté s jistotou zaútočit na produkční verzi.

Již jsme se jednou zmínili, že je zapotřebí znát dva údaje pro prolomení účtu – login a heslo. Jsme opět v situaci, abychom zbytečně loginy nevystavovali veřejnosti. Při klasickém přihlášení do WP můžeme vidět dvě hlášky. První nastává, pokud je login chybný:

```
CHYBA: Neplatné uživatelské jméno.
Zapomněli jste heslo?
```

A druhá hláška nastává, pokud login je správný, ale heslo je chybné:

```
CHYBA: Chybně zadané heslo pro uživatelské jméno LOGIN.
Zapomněli jste heslo?
```

Z tohoto rozdílu hlášek lze vypozorovat, zda daný login existuje či ne. Nejlepší řešení je, pokud se na obě chyby vrací stejné hlášky. Tyto hlášky lze změnit pomocí `login_errors` filtru.

4.2.4 Přístupová práva k souborům

Přístupová práva k souborům nám říkají, kdo může číst/měnit soubory a vstupovat do složek. Rozhodně nechceme dát příliš štědrá přístupová práva veřejnosti – pročítání souborů či spouštění určitých souborů může vést k narušení bezpečnosti.

Uveďme si nejprve obecná doporučení pro WordPress instalaci [13]:

1. Všechny soubory/složky by měly být ve vlastnictví skutečného uživatele, ne účtu, který využívá httpd proces.

2. Vlastník skupiny je irelevantní (mohou být výjimky).

3. Všechny složky by měly mít oprávnění 755 nebo 750.

4. Všechny soubory by měly mít oprávnění 644 nebo 640 s výjimkou `wp-config.php`, který by měl mít 400.

5. Žádný adresář by neměl mít 777.

Uvedená oprávnění jsou tzv. „tradiční unixové oprávnění“. Podrobnější vysvětlení lze najít na adrese `https://en.wikipedia.org/wiki/File_system_permissions`.

1. a 2. bod je relevantní, pokud provozujeme vlastní webový server. Jelikož využíváme služeb třetí strany, nemůžeme tyto aspekty ovlivnit. Ve všech bodech je použit podmiňovací způsob, protože lze narazit na situace, kdy je potřeba změnit tyto oprávnění.

4.3 Zabezpečení WordPress

Jelikož je WordPress velice rozšířený systém, existuje již celá řada řešení pro zabezpečení. Tato řešení lze stáhnout formou pluginu (placené či zdarma). Pro přiblížení funkcí takových pluginů si vyberu jeden plugin a popíši funkce a mechanismy, které využívá.

Zvolil jsem plugin Wordfence Security, který se řadí k jedním z nejlepších pluginů na zabezpečení, verze zdarma není tolik omezená, vývoj pluginu je stále aktivní a hlavně má mnohonásobně víc aktivních instalací[11] oproti ostatním pluginům.

Využiji pouze shrnutí funkcí tohoto pluginu. Ke každé funkcionalitě uvedu, jak ji lze efektivně řešit, ale nebudu popisovat implementaci v tomto pluginu.

4.3.1 Wordfence Security – Firewall & Malware Scan

Hlavním obsahem tohoto pluginu je firewall a skener mallwaru. Popišme si je postupně. [14]

4.3.1.1 Blokování malicious traffic

Malicious traffic je veškerá komunikace, která není standardní komunikace s webovým serverem – tj. používání stránek uživateli, volání cronu pro údržbu, indexování stránek pro komunikaci apod. Malicious traffic jsou zejména skenery, spamy (popř. backscatter[12]) a DOS útoky.

Tyto „útoky“ lze odhalit na základě chování komunikace. Např. typické chování scannerů je, že přichází plno požadavků na různé stránky či subdomény[13]. Ochrana tedy spočívá v naučení se vzoru chování komunikace a poté hledat změny komunikace oproti naučenému vzoru. Komunikaci, která se vymyká normálnímu chování lze označit za podezřelou a aplikovat pokročilejší rozpoznávací metody a až poté komunikaci zablokovat či povolit.

[11] Počet aktivních instalací na začátku dubna 2019 je podle wordpress.org přes 3 miliony.

[12] Spam Backscatter je nežádoucí efekt nedoručenky poštovního serveru – odpověď poštovního serveru, že spam nebyl doručen (to je samozřejmě taky spam).

[13] Přednostně se zde bavíme o webových stránkách; Scanner celého serveru by mohl zkoušet různé porty na stejném cíli.

Tento plugin se po instalaci nejprve nastaví do learning módu, kde po několik dnů se učí vzor chování. Teprve potom se doporučuje zapnout firewall.

4.3.1.2 Malware skener pro požadavky na server

Další funkce je kontrolování příchozích dat na přítomnost škodlivého kódu či obsahu v požadavcích na server. Tato funkce nám pokrývá kapitolu 3. V každém případě toto je pouze dodatečné zabezpečení a nenahradí řádnou validaci vstupních dat v našem kódu.

4.3.1.3 Ochrana proti brute force útoku

Na webové stránce by vždy měla být nějaká implementace proti online útoku hrubou silou[14].

Ochrana se dá řešit limitování počtu pokusů přihlášení. Po dosažení maximálního počtu pokusů přihlášení je vhodné nastavit minimální čas, kdy se lze znova pokusit přihlásit. Dle mého zablokovat účet je pro nás špatné, protože dáváme útočníkovi jednoduchou možnost, jak provést DoS útok – může zablokovat všechny účty. Také lze nastavit přísnější politiku síly hesla nebo zavést vícefaktorové přihlášení.

Dále, pro automatizované útoky na webové stránky, lze vytvořit alternativní URL adresu pro přihlašování. Výchozí stránka pro přihlášení do WP je `/wp-login.php`, kde probíhá přihlášení. Automatické útoky budou cílit právě sem. Tato stránka, mimo jiné, útočníky rovněž informuje, že se jedná o web postavený na WordPressu. Pokud tuto ochranu chceme implementovat, musíme rovněž zajistit, aby jsme vypnuli i přesměrování na přihlášení (např. pokud bez přihlášení chceme vstoupit do administrace).

MoveAndFight nepřihlášeným uživatelům vůbec nedovolujeme vstoupit na tuto stránku. Pro nepřihlášené uživatele je vytvořen white list stránek (tj. seznam povolených stránek).

4.3.1.4 Kontrola zdrojových kódu WP, šablony a pluginů

Jedna ze základních kontrol je porovnání zdrojových kódů samotného WordPressu vůči oficiálnímu repozitáři. Toto odhalí, pokud útočník skryl nějaký kód právě do těchto souborů. Zde lze ukrýt „zadní vrátka“ pro autorizované akce, odesílání citlivých dat na jiný server apod. Pokud by byl nějaký soubor pozměněn, tak právě tento skener by okamžitě zahlásil riziko s kritickou úrovní.

[14] Online brute force – útok probíhá přes požadavky na server,
Offline brute force – prolomení hesla probíhá lokálně u útočníka (k tomuto by nikdy nemělo dojít, útočník nesmí získat informace, které by mohl u sebe luštit).

Podobně se dají kontrolovat i šablony. Toto samozřejmě lze jen, pokud máme šablonu z oficiální WordPress theme repozitáře. Plugin kontroluje soubory přímo s používanou verzí na webu. Navíc kontroluje všechny nainstalované šablony a ne jen aktivní. Samozřejmě, pokud někdo šablonu upraví pro potřeby webových stránek, pak tato kontrola je zbytečná.

Úplně stejně jako šablony se kontrolují i pluginy. Zde je výhoda, že pluginy se méně často upravují pro potřeby webu. Úskalím je, pokud autor pluginu nerespektuje oficiální postup pro aktualizaci pluginu. To poté může vést k nesprávnému označení rizika. Proto před jakoukoliv akcí je nutné zkontrolovat změny, které jsou v pluginu provedeny.

4.3.1.5 Kontrola ostatních souborů kvůli virům, podezřelým kódům a odkazování/přesměrování

Pokud se nekontrolují známe soubory (jako jsou zdrojové kódy WP atd.), tak se provádí hloubková kontrola, při které se testuje přítomnost kódů oproti databázi škodlivých kódů.

Útočník může škodlivý kód ukrýt pomocí kódování (např. base64, URL endcode či kódování v hexadecimální soustavě). I tyto zakódované části jsou prohledány.

V souborech se zároveň hledají i přesměrování, které by mohly uživatele zavést do nebezpečných míst internetu. Nekontroluje se pouze přesměrování, ale i odkazy, které by mohly stáhnout škodlivý obsah.

4.3.1.6 Skenování příspěvků a komentářů

Jelikož uživatelé smí přidávat jak příspěvky, tak i komentáře, je potřeba kontrolovat i tento obsah. Na rozdíl od souborů jsou tyto data uložená v databázi. Tato kontrola se kvůli rychlosti provede přímo přes databázový server (a ne přes WP funkce). Obsah je testován opět na škodlivý kód a odkazy. Vždy se kontrolují všechna data a ne jen nově přidané.

4.3.1.7 Kontrola nastavení WP, šablon a pluginů

I v samotném nastavení stránek, které je uloženo v databázi, může být škodlivý obsah. Toto je kontrolováno úplně stejně jako příspěvky a komentáře.

4.3.1.8 Ověření aktuálních verzí

Z předchozích kapitol víme, že je potřeba udržovat instalace aktuální. Skener odhalí neaktualizované části projektu (opět samotný WP, šablony a pluginy) a do výsledku skenu vypíše seznam komponent k aktualizaci.

4.3.1.9 Další kontroly

Skener kontroluje mnoho dalších dat a nastavení. Např. zda nebyl admin vytvořen mimo WP, zda nedochází místo na disku, zda nedošlo k neautorizované změně v DNS záznamech. Kontroluje soubory mimo WP a binární soubory.

Jak je vidět, kontrola může odhalit spoustu potenciálních nebezpečí. Test na MoveAndFight odhalil nutnost aktualizovat 6 pluginů a jádro WP. Dále odhalil soubor, který do WP nepatřil.

4.3.2 Salt Shaker

Tímto pluginem bych rád zmínil další zabezpečení WP, které nám pomáhá zabezpečit cookie (zabrání i Session hijacking – kapitola 3.4). WP všechny cookie, které využívá šifruje. Sůl a klíč pro šifrování lze najít v `wp-config.php`.

Tyto hodnoty ve `wp-config.php` si lze vygenerovat přímo na stránkách WP – `https://api.wordpress.org/secret-key/1.1/salt/`. Náhradou stávajících klíčů má za následek násilné odhlášení všech uživatelů. Toto zároveň odstraní riziko, pokud někdo získal naše klíče pro cookie. Stejně tak přestanou platit podstrčené či odposlechnuté cookie data včetně SID.

Tento plugin dokáže změnu klíčů pro cookie data zautomatizovat na periodické bázi (denní, týdenní a měsíční) nebo vynutit okamžitou změnu. Denní změna klíčů je u uživatelů už spíš na obtíž, dle mého měsíční obnova je dostačující.

4.3.3 Záloha

Poslední důležitou částí bezpečnosti jsou zálohy webu. Je potřeba periodicky zálohovat data v databázi a uživatelské soubory (soubory WP/pluginů lze znova získat z repozitáře WP a šablonu máme ve svém repozitáři, který je chráněn). Zálohovat chceme, protože vždy hrozí riziko smazání důležitých dat, ať už v důsledku útoku, tak i nechtěné smazání (chybným kódem, autorizovanou osobou, …).

Pro zajímavost se podívejme na možnosti záloh. První možnost je provádět vlastní zálohu – patrně instalací pluginu, který nám data uloží na předem definované místo. Další možností je záloha z pohledu serveru. Tato záloha nebere výpočetní výkon našemu hostingu. Podívejme se, jaké zálohy provádí náš poskytovatel Wedos s.r.o.

V naší variantě NoLimit se záloha vždy provádí jednou týdně pro potřeby poskytovatele. Tuto obnovu mohou poskytnout na požádání do 7 dnů za jednorázový poplatek 605 Kč vč. DPH. Platíme tedy pouze, pokud je potřeba

obnova a dostaneme kopii souborů až 7 dní starou. Obnova dat zpět je zcela v naší režii.

Další možností je využít lepší službu NoLimit Extra, kde se zálohuje denně (soubory, databáze i emaily), lze si vybrat z posledních pěti záloh a obnova je zdarma. Samotná služba je ale dražší o 91 Kč měsíčně (samozřejmě v cenně jsou vyšší limity na paměť, SNI „zdarma“ a větší uložiště). Za rok po odečtení příplatkových služeb to činí rozdíl 945 Kč.

Pokud bychom zálohu řešili pomocí pluginu, tak se podívejme na požadované funkcionality:

- automatická záloha dat WP (nejlépe denní báze),
- uložení zálohy mimo webhosting (kvůli bezpečnosti a navíc Wedos nedovoluje ukládat zálohy do uložiště pro webové stránky)
- možnost rozšíření zálohy na soubory a tabulky, které nejsou WP

Podívejme se na několik pluginů a porovnejme si je.

4.3.3.1 UpdraftPlus WordPress Backup Plugin

Verze zdarma podporuje zálohu WP souborů a WP tabulek, které je možno uložit na Dropbox, Google Drive, Amazon S3, UpdraftVault (uložiště samotného pluginu), Rackspace Cloud, FTP, DreamObjects, Openstack Swift a email. Zálohy lze plánovat, prohlížet si zazálohovaný obsah a jedním klikem data obnovit.

Placená verze je o poznání bohatší. K již uvedeným úložištím zvládá navíc Microsoft OneDrive, Google Cloud, SFTP a další. Důležitou částí je i záloha tabulek a dat, které nejsou součástí WordPressu. Z bezpečnostního hlediska umí zálohu databáze zašifrovat a využít šifrování v FTP. K placené verzi pluginu rovněž nabízí 1 GB místa na svém uložišti UpdraftVault. Cena této služby je po přepočtu 1 708 Kč za první rok a 1 025 Kč za každý další. [15]

Tento plugin aktivně využívá přes 2 miliony webů.

4.3.3.2 Jetpack od WordPress.com

Tento plugin se nestará pouze o zálohu webu, ale poskytuje i ochranu proti spamu, ochranu proti brute force útoku, SEO optimalizaci, tlačítko pro placení přes PayPal, inzertní program atd. Důležité je, že verze zdarma zde vůbec neobsahuje možnost zálohy, proto se rovnou podívejme na placenou verzi `Personal`.

Verze `Personal` podporuje denní zálohu, kterou ukládá na své servery (neomezené uložiště pro poslední měsíc). V databázi zálohuje všechny tabulky

začínající prefixem definovaném ve `wp-config.php` (tzn., že by se v našem případě musely přejmenovat tabulky). Tato verze obsahuje všechny funkce pro bezpečnost, které nabízí, kromě funkce Malware scanning. Cena je 936 Kč za rok. Počet aktivních instalací je nad 5 miliónů. [16]

4.3.3.3 BackupBuddy

Tento plugin má pouze placenou verzi, za kterou se získá plugin a roční členství pro aktualizace a podporu.

Největší výhoda tohoto pluginu je, že nám dovolí přímo vybrat dodatečné tabulky (dokonce i další databáze) pro zálohu. Rovněž můžeme vybrat složky, které má zálohovat. Záloha se může ukládat na BackupBuddy, kde dostaneme 1 GB prostoru. Dále podporuje Dropbox, Email, FTP, Amazon S3, Google Drive atd. Automatickou zálohu lze nastavit až na hodinovou bázi.

Cena tohoto pluginu činí 1 828 Kč ročně. Pokud nepotřebujeme plně aktuální verzi, tak není potřeba platit další rok. Plugin nám už zůstane. [17]

Kapitola 5

Implementace zabezpečení

Nyní se podívejme na samotné zabezpečení aplikace MoveAndFight. Pro začátek si shrňme, co je třeba zabezpečit, v jakém stavu je aplikace a která zabezpečení jsme již provedli.

Z kapitoly 2 víme, že je web potřeba verzovat projekt, šifrovat komunikaci a mít testovací verzi. Přímo v teoretické části, jsme si řekli, jak je možné tyto záležitosti obstarat. Verzujeme v Gitu, šifrujeme pomocí vynucením HTTPS přes htaccess a testovací verzi máme na `https://test.moveandfight.cz`. Je vhodné mít alespoň nějaký implementační postup – s větším projektem rostou nároky na implementační postup a míru testování. V našem menším projektu si vystačíme s řešením uvedeném ve zmíněné kapitole, tedy přes Git. Jelikož jsou tyto kroky popsané v teoretické části nebo nejsou stěžejní částí této práce vynechám je zde.

V kapitole 3 jsme se dozvěděli, proč je nutné správně validovat a vypisovat data, proto musíme řádně nastudovat stávající implementaci, kterou zabezpečíme.

V poslední teoretické kapitole 4 jsme se naučili, jak správně nastavit WP. Ukážeme si kód, se kterým nastavím role a oprávnění. Jelikož dodatečná zabezpečení jsou řešena nainstalování a správným nastavením pluginu, rovněž vynechám. Správné nastavení lze vyhledat v dokumentaci daného pluginu.

V této části budou ukázky použitého kódu. Kód je pro účely dokumentu upraven a zkrácen. Na první řádku bude vždy komentář s umístěním souboru na přiloženém disku. Pokud komentář nebude přítomen, jedná se o kód čistě pro ukázku a není součástí odevzdaných zdrojových kódů.

5.1 Validace vstupů

Formuláře MoveAndFight jsou řešené pomocí mého pluginu FormPlugin[15]. Tento plugin se snaží o možnost recyklovat jednotlivé formuláře a tak dát možnost mít stejný formulář na vícero stránkách (dá se říct, že je to primitivní šablonovací nástroj pouze pro formuláře).

Každý formulář se skládá ze dvou funkcí, které jsem nazval `echo` a `save`. V `echo` funkci se nalézá HTML formulář a `save` obsahuje script, který se provede po odeslání formuláře. Zde může být samotná validace a samozřejmě změny provedené odesláním formuláře. Momentálně se v žádném formuláři nenalézá kontrola vstupu.

Jak bylo v teoretické části zmíněno, validace je velmi často stejná pro mnoho vstupů, proto tento proces chceme automatizovat a kontroly vstupů přiřazovat. Vytvořme si tedy základní kontroly, které budeme používat. Tento soubor může vypadat takto:

Výpis kódu 5.1: FormPlugin – Definování kontrol

```
1  <?php // /sources/formplugin/Namespaces/terms.php
2  namespace FormPluginTerms;
3
4  //Kontrola datového typu / významu
5  function type($value, $param){
6      switch($param){
7          case("string"): return is_string($value);
8          [...]
9          case("email"): return
10             filter_var($value, FILTER_VALIDATE_EMAIL)
11             ? true : false;
12     }
13     return null;
14 }
15
16 //Zda je pole povinné
17 function requied($value, $param = true){
18     if($param === false) return true;
19     return $value !== null;
20 }
21
22 //Zda je to číslo a zda je vetší rovno parametru
23 function min_val($value, $param){
24     return type($value, "numeric") && $value >= $param;
25 }
```

Toto jsou velice jednoduché kontroly. Složitější kontroly mohou být například na validaci nahraného soubory apod.

[15]Plugin je veřejný a dostupný na adrese `https://Bagnat@bitbucket.org/bagnatcz/formplugin.git`

Nyní potřebujeme pluginu sdělit, na který vstup má aplikovat dané filtry, proto ke každému formuláři definujeme požadavky pomocí nepovinné funkce umístěné v souboru s formulářem. Například definice kontrol během registrace může vypadat takto:

Výpis kódu 5.2: FormPlugin – Přiřazení kontrol k formuláři

```
<?php // /sources/forms/form-registrace.php
function form_registrace_terms()
{
    return array(
        "nick" => array(            //název vstupu
            "requied"   => true,    //zda je vyžadován
            "min_len"   => 3,       //minimální délka
            "type"      => "string" //datový typ
        ),
        "mail" => array(
            "min_len"   => 6,
            "type"      => "email"
        ),
        "passwd" => array(
            "type"      => "array",
            "count"     => 2,       // pole o 2 prvcích
            "equal"     => true     // itemy pole se musí rovnat
        ),
        [...]
    );
}
```

Kontrola, zda heslo splňuje politiku vytváření hesla aplikace lze dodat přímo ve formuláři, či dodělat další datový typ `password`.

Nyní samozřejmě musíme udělat script, který vše správně zkontroluje a v případě chyby nám vrátí seznam nesplněných kritérií. Tento script lze vytvořit takto:

Výpis kódu 5.3: FormPlugin – Kontrola formuláře

```
<?php // /sources/formplugin/Classes/Form.php, řádky 98-120

$errors = array();
if( $this->podminky ) //pokud jsou nějaké podmínky přiřazené
{
    foreach($this->podminky as $name => $detail){
        // $name    = název vstupu
        // $detail  = pole podmínek

        //není povinný a nemá další omezení
        if( [/*je nepovinný AND není nastavený*/]) continue;

        foreach ($detail as $function => $param){
            // $function   = term (např.: 'type')
            // $param      = hodnota (např.: 'string')

            $funkce = "FormPluginTerms\\".$function;
            if(function_exists( $funkce )){
                if( ! $funkce( $input[$name], $param) )
                    $errors[$name][] = $function;
            }else{
                if(!isset($errors["terms"]))
                    $errors["terms"] = array();
                $errors["terms"][] = $function;
            }
        }
    }
}
```

Proměnná `$this->podminky` obsahuje pole definované ve funkci `form_{form-name}_terms`. Do pole `$errors` se ukládají všechny chyby, které vstup nesplnil. Pokud například u jména nebude splněna délka vstupu bude pole `$errors` obsahovat `array( "name" => array( "min_len" ) )`.

Kontrolu, zda je formulář správně vyplněný lze tedy jednoduše v `save` funkci zapsat takto:

Výpis kódu 5.4: FormPlugin – Save funkce – kontroly

```
<?php // /sources/forms/form-registrace.php, řádky 88-137

function form_registrace_save($data, $errors)
{
    if($errors) {
        // ošetření chyb
    }else {
        // dodatečné kontroly
        // samotná registrace
    }
}
```

Jelikož všechny formuláře na MoveAndFight již jsou v tomto pluginu, stačí pouze dodefinovat podmínky u každého formuláře a provést refaktoring skriptu pro ukládání.

5.2 Bezpečné ukládání vstupů

Nyní se podívejme na druhou část, tedy na správné ukládání/vypisování vstupů. Každý uživatelský textový vstup musí být překonvertován pomocí funkce `htmlspeacialchars`. Opět využijeme našeho pluginu a tuto konverzi provedeme ihned po přijetí vstupu. Opět chceme celý proces udělat automaticky, neměnit rozhraní pluginu a snažíme se, o co nejuniverzálnější řešení (možnost aplikovat i jiné funkce na vstup). Přidáme tedy další rozšíření – filtry.

Opět si definujme požadované filtry:

Výpis kódu 5.5: FormPlugin – Definování filtrů

```
<?php // /sources/formplugin/Namespaces/filters.php
namespace FormPluginFilters;

//Upraví text na povolený obsah
function user_text_content($value){
    return htmlspecialchars($value);
}

//Upraví pole na povolený obsah
function user_text_content_array($value){
    foreach ($value as &$v) $v = htmlspecialchars($v);
    return $value;
}
```

Poté definovat u každého formuláře, kde se mají filtry aplikovat:

Výpis kódu 5.6: FormPlugin – Přiřazení filtrů k formuláři

```
<?php // /sources/forms/form-registrace, řádky 176-184
function form_main_filters()
{
    return array(
        "nick"      => array("user_text_content"),
        "country"   => array("user_text_content"),
        "firstname" => array("user_text_content"),
        "lastname"  => array("user_text_content"),
    );
}
```

Script, který filtry aplikuje, se spustí okamžitě po kontrole vstupu. Na vstup, který není nastavený, má null hodnotu nebo nesplnil nějaký požadavek, filtr neaplikuje. Lze kód změnit, aby se filtr aplikoval vždy, poté by se daly filtry využít, aby například dodaly výchozí hodnoty. V tomto případě by sám filtr musel alespoň testovat přítomnost hodnoty.

Kód, který aplikuje filtry:

Výpis kódu 5.7: FormPlugin – Save funkce – filtry

```
<?php // /sources/formplugin/Classes/Form.php, řádky 122-149

if( $this->filtry ) {  //pokud jsou nějaké filtry přiřazené

   foreach($this->filtry as $name => $filters) {

      //filtr se provede, pouze pokud vstup obstál u všech podmínek
      if( !isset($errors[ $name ]) ){

         //není nastavený nebo povinný
         if( !isset($input[$name]) || $input[$name] === null)
            continue;

         foreach ($filters as $filtr)
         {
            $funkce = "FormPluginFilters\\".$filtr;
            if( function_exists( $funkce ) )
               $input[$name] = $funkce( $input[$name] );
         }

      }else
         $errors[$name]["filters"] = true;
   }
}
```

V případě, že filtr neexistuje, dostaneme chybu opět v `$errors`.

5.3 SQL injection

Webová aplikace MoveAndFight je odolná vůči tomuto útoku, protože již od základů využívá správně rozhraní pro databáze. Krom používání třídy `wpdb` využívá také třídu `PDO`, kde bezpečnost je řešena podobně. Data se rovněž vystavují zvlášť a třída samotná se stará o nahrazování samostatných znaků.

5.4 Nastavení rolí a oprávnění

K nastavení rolí je třeba znát, které role budou potřeba. Máme zde administrátora, který může vše. Tento post bude pouze vývojář a jako jediný může aktualizovat WP/pluginy (zde hrozí potenciální riziko nekompatibility s předcházející verzí) a provádět podobné nebezpečné akce.

Samozřejmě zde potřebujeme mít uživatele, kteří budou oprávněni měnit obsah stránek přímo z administrace. Tento post budou zastávat manažeři. Manažer bude moci vytvářet/měnit stránky, spravovat obsah (tj. vytváření turnajů, schvalování úkolů od hráčů, odpovídat na dotazy z podpory apod.).

Speciální kategorií lidí budou redaktoři. Redaktor má za úkol pouze vytvářet články (o zdraví a cvičení), které nebude moc sám vydat. Toto vytváření bude samozřejmě přes rozhraní WP.

Poslední kategorií bude normální uživatel, kterého nazveme hráč. Hráč nepotřebuje žádná práva, protože každý úkon bude provádět script pluginu nebo šablony, takže nebude vůbec využívat rozhraní WP. Toto rozhraní si nebude moc ani zobrazit.

Jako první musíme odstranit nepotřebné role a následně přidat potřebné.

Výpis kódu 5.8: Odebrání a přidání rolí

```
1  <?php // /sources/rbac_setup.php, řádky 10-24
2
3  // odebrání rolí
4  $odebrat = array('editor', 'author', 'contributor',
5           'subscriber', 'manager', 'redaktor', 'hrac');
6  foreach ($odebrat as $role){
7     if( get_role($role) ){
8        remove_role( $role );
9     }
10 }
11
12 // přidání rolí
13 $pridat = array('manager' => "Manažer", 'redaktor' => "Redaktor",
14          'hrac' => "Hráč");
15 foreach ($pridat as $slug => $name){
16    add_role( $slug, __( $name ) );
17 }
```

Každá role má unikátní identifikátor – slug. Dále má rovněž název, který se vypisuje v administraci. Funkce `__( string )` slouží pro překlad webu. Jelikož nemáme definované překlady, tak se tato funkce chová jako identita.

Odebíráme rovněž i nové role. Pokud totiž později budeme chtít odebrat některá oprávnění roli, musíme explicitně říct, že oprávnění chceme odebrat. Pokud ale odebereme celou roli, odeberou se i všechna oprávnění. Následně provedeme opětovné přidání oprávnění, které nyní nastavíme. Pokud už uživatelé mají přiřazenou roli, kterou odebereme a následně přidáme, role mu zůstane.

Nyní můžeme přidat oprávnění jednotlivým rolím. Jelikož jsem administrátora nikdy neodebral, zůstávají mu všechna oprávnění WordPress. Každopádně nová oprávnění ještě přidané nemá. Jedná se o funkce, které jsou přidané s novými typy článků. S těmito články může pracovat i manažer, proto je přidáme zároveň.

Výpis kódu 5.9: Přidání oprávnění adminovi a manažerovi

```
1  <?php // /sources/rbac_setup.php, řádky 30-44
2
3  global $wp_roles; // global class wp-includes/capabilities.php
4  $opravneni = array(
5     'edit_{slug}', 'read_{slug}', 'delete_{slug}',
6     'edit_{slug}s', 'edit_others_{slug}s',
7     'publish_{slug}s', 'read_private_{slug}s', 'edit_{slug}s'
8  );
9
10 $roles = array('administrator', 'manager');
11
12 $typyClanku = array("cvik", "ukol", "doplnek",
13              "turnaj", "legie", "clanek", "post");
14
15 foreach ($roles as $role) {
16    foreach ($typyClanku as $typ){
17       foreach ($opravneni as $name){
18          $cap = str_replace("{slug}", $typ, $name);
19          $wp_roles->add_cap( $role, $cap, true );
20       }
21    }
22 }
```

Proměnná `$opravneni` obsahuje seznam oprávnění k jednomu typu článku, kde řetězec `{slug}` značí název typu článku. `$roles` obsahuje seznam rolí, ke kterým budeme oprávnění přiřazovat. Proměnná `$typyClanku` jsou všechny nové typy článků. Právě tyto názvy jsou slugy článků.

Jak je vidět, Zavoláme pro každou roli, pro každý typ článku a pro každé oprávnění metodu `add_cap`, která přidá oprávnění k roli. Tímto jsme dovolili administrátorovi a manažerovi měnit obsah webu.

Ještě nesmíme zapomenout přidat oprávnění ke změnám v taxonomiích. Taxonomie v redakčním systému WordPress jsou tagy nebo kategorie článků[16]. Například články typu Doplněk obsahuje kategorie (např. „Proteiny“, „Sacharidy“ apod.). I k těmto kategoriím je třeba přidat oprávnění. To lze udělat pomocí tohoto skriptu:

Výpis kódu 5.10: Přidání oprávnění k taxonomy

```
<?php // /sources/rbac_setup.php, řádky 68-77

$roles = array('administrator', 'manager');
$typyTaxonomy = array("clanek_tag", "clanek_kategorie",
        "cvik_kategorie", "doplnek_kategorie");
$opravneni = "manage_categories_{slug}";

foreach ($roles as $role) {
   foreach ($typyTaxonomy as $typ){
      $cap = str_replace("{slug}", $typ, $opravneni);
      $wp_roles->add_cap( $role, $cap, true );
   }
}
```

`$typyTaxonomy` obsahuje názvy (slugy) kategorií a tagů přidaných taxonomií. Tyto názvy jsou definované při registraci taxonomie.

Oprávnění `manage_categories_{slug}` dovoluje kompletně spravovat taxonomie. Toto, ale není build-in oprávnění WP, kdežto oprávnění definované přímo při registraci taxonomie. Jelikož se oprávnění definují úplně stejně i při registraci nového typu článku, ukažme si, jak taková registrace vypadá.

Výpis kódu 5.11: Registrace taxonomie

```
<?php
$labels = array(
  'name'  => __( 'Kategorie' ),  [...]
);

$args = array(
  'labels'        => $labels, [...],
  'capabilities'  => array(
    'manage_terms'  => 'manage_categories_doplnek_kategorie',
    'edit_terms'    => 'manage_categories_doplnek_kategorie',
    'delete_terms'  => 'manage_categories_doplnek_kategorie',
    'assign_terms'  => 'manage_categories_doplnek_kategorie',
  )
);

register_taxonomy( 'kategorie_doplnku', 'doplnek' , $args );
```

[16]Název kategorie/tagu je ve WP značen term (tabulka wp_term), kdežto jednotlivé názvy kategorií/tagů jsou taxonomie (tabulka wp_term_taxonomy); propojení taxonomie a článku je vztah (tabulka wp_term_relationships).

Pro ulehčení práce jsem všem oprávněním přiřadil pouze jedno oprávnění. Není v plánu dávat pouze část přístupových práv k taxonomiím.

V poslední řadě nastavíme dodatečná oprávnění. Mezi nimi jsou speciální stránky pro manažera, povolení redaktorovi přístup do administrace WP atd. Povolení redaktorovi psát články je nastaveno stejně jako v příkladě kódu 5.9.

Výpis kódu 5.12: Přidání dalších oprávnění

```
1  <?php // /sources/rbac_setup.php, řádky 80-94
2
3  $pravidla = array(
4    'edit_users'            => array('administrator', 'manager'),
5    'read'                  => array('administrator', 'manager',
6                                'redaktor'),
7    'admin_chyby'           => array('administrator'),
8    'admin_kvizy'           => array('administrator', 'manager'),
9    'admin_trener'          => array('administrator', 'manager'),
10   'admin_propojeni_ukolu' => array('administrator', 'manager'),
11   'admin_mail'            => array('administrator', 'manager')
12 );
13
14 foreach ($pravidla as $cap => $roles){
15   foreach ($roles as $role){
16     // $role   -> název role
17     // $cap    -> název oprávnění
18     $wp_roles->add_cap( $role, $cap, true );
19   }
20 }
```

Tímto jsem nastavili kompletně role a oprávnění. Nyní stačí skript jednou spustit a otestovat, zda vše funguje jak očekáváme. Pokud chceme zkontrolovat přiřazená oprávnění, stačí si vypsat pole, které vrátí skript

`get_role( 'administrator' )->capabilities`.

Závěr

Cílem teoretické práce bylo vyzdvihnout nebezpečné a běžné útoky na webovou aplikaci. Výsledek teoretických rešerší je důležitý, protože se v nich poukázalo na důležité aspekty bezpečnosti.

Důležitá část zabezpečení, kterou může programátor ovlivnit, je validace vstupů od uživatele a šifrování spojení. Právě toto zabezpečení znemožní nejběžnější útoky na webové stránky.

V praktické části jsme implementovali všechny body zadání. Pomocí služeb webového serveru jsme zajistili šifrování a vynucení bezpečného protokolu pro komunikaci. Vlastní úpravou šablony a vytvořením pluginu pro kontrolu vstupů jsme pokryli základy zabezpečení webového projektu a instalací vhodných řešení třetích stran jsme implementovali pokročilé ochrany stránek.

Do budoucna by se dala práce rozšířit o další úrovně zabezpečení a odolnosti vůči chybám, které nesouvisí tolik s útoky, ale se správným navrhováním skriptů, databáze[17] a automatické testování funkcionalit webových stránek.

[17]Zde mám na mysli správně vytvářet omezení (constraints), aby nevznikaly nekonzistentní záznamy, efektivnější struktura tabulek apod.

Kapitola 6

Seznam použitých zkratek

CA Certifikační autorita

CORS Cross-origin resource sharing

DoS Denial of Service

dtb databáze

HTML Hypertext Markup Language

HTTP(S) Hypertext Transfer Protocol (Secure)

JS JavaScript

MAF MoveAndFight (název projektu)

OS Operační systém

PDO PHP Data Objects

PHP Hypertext Preprocessor

PKI Public Key Infrastructure

RBAC Role-based access control

SID Session ID

TLD Top level domain (např. „.cz“ nebo „.com“)

TLS Transport Layer Security

URL Uniform Resource Locator

WP WordPress

XSS Cross-site scripting

Kapitola 7

Obsah přiloženého CD

```
/
├── readme.md ..... stručný popis obsahu CD
├── thesis.pdf ..... text práce ve formátu PDF
├── documentation ..... zdrojová forma práce ve formátu LaTeX
│   └── sources ..... ukázky z této práce členěné podle kapitol
└── sources
    ├── formplugin ..... plugin pro práci s formuláři ve WP
    ├── forms ..... ukázka nastavení několika formulářů pro MAF
    └── rbac_setup.php ..... nastavení rolí a oprávnění pro MAF
```

Obsah FormPlugin složky

```
/sources/formplugin/
├── readme.txt ..... stručný popis FormPluginu pro repozitář WP
├── FormPlugin.php ..... inicializační soubor pluginu
├── Classes ..... složka pro třídy pluginu
│   └── Form.php ..... třída Form
├── Examples ..... složka pro ukázky použití pluginu
│   └── form-example.php ..... ukázka použití pluginu
└── Namespaces ..... složka s definicemi funkcí pro plugin
    ├── errors.php ..... chybové funkce
    ├── filters.php ..... filtry pro uživatelský vstup
    └── terms.php ..... funkce pro kontrolu uživatelského vstupu
```

Bibliografie

1. CHACON, Scott; STRAUB, Ben. *Pro git.* Apress, 2014.
2. WEDOS INTERNET, a.s. *Webhosting - subdomény* [online]. 2010-09-20, 2010 [cit. 2019-04-12]. Dostupné z: `https://kb.wedos.com/cs/webhosting/subdomeny.html`.
3. WORDPRESS.ORG. *Changing The Site URL* [online]. 2019 [cit. 2019-04-12]. Dostupné z: `https://codex.wordpress.org/Changing_The_Site_URL`.
4. NAZIRIDIS, Nick. *Browsers and Certificate Validation* [online]. 2018 [cit. 2019-05-06]. Dostupné z: `https://www.ssl.com/article/browsers-and-certificate-validation/`.
5. WEB SECURITY SOLUTIONS LLC. *SSL Certificate Providers Comparison: The Best SSL Provider* [online]. 2019 [cit. 2019-04-12]. Dostupné z: `https://cheapsslsecurity.com/sslcompare/ssl-certificate-providers-comparison.html`.
6. LET'S ENCRYPT. *About us* [online]. 2017 [cit. 2019-04-12]. Dostupné z: `https://letsencrypt.org/about/`.
7. WICHERS, Dave. *Types of Cross-Site Scripting* [online]. 2017 [cit. 2019-05-06]. Dostupné z: `https://www.owasp.org/index.php/Types_of_Cross-Site_Scripting`.
8. OWASP FOUNDATION. *Session hijacking attack* [online]. 2014 [cit. 2019-04-18]. Dostupné z: `https://www.owasp.org/index.php/Session_hijacking_attack`.
9. WORLD WIDE WEB CONSORTIUM. *HTML 5.2: W3C Recommendation* [Cookie setter (http-equiv="set-cookie")] [online]. 2017 [cit. 2019-04-18]. Dostupné z: `https://www.w3.org/TR/html52/document-metadata.html#statedef-http-equiv-set-cookie`.

10. CHROMESTATUS.COM. *<meta http-equiv="set-cookie" ...> (removed)* [online]. 2018 [cit. 2019-04-18]. Dostupné z: `https://www.chromestatus.com/feature/6170540112871424`.

11. MAUNDER, Mark. *WordPress Table Prefix: Changing It Does Nothing to Improve Security* [online]. 2016 [cit. 2019-04-26]. Dostupné z: `https://www.wordfence.com/blog/2016/12/wordpress-table-prefix/`.

12. WEDOS INTERNET, a.s. *htaccess - zobrazení seznamu souborů v adresáři* [online]. 2011 [cit. 2019-04-18]. Dostupné z: `https://kb.wedos.com/cs/htaccess/seznam-souboru.html`.

13. PODTECH.IO. *Correct file permissions for WordPress* [online]. Ed. ADIGA. 2017 [cit. 2019-04-18]. Dostupné z: `https://stackoverflow.com/a/47571424`.

14. WORDFENCE.COM. *Scan Options - Wordfence* [online]. 2019 [cit. 2019-04-18]. Dostupné z: `https://www.wordfence.com/help/scan/options/`.

15. UPDRAFTPLUS. *Comparison of UpdraftPlus free and UpdraftPremium* [online]. 2019 [cit. 2019-05-11]. Dostupné z: `https://updraftplus.com/comparison-updraftplus-free-updraftplus-premium/`.

16. INC., WordPress.com / Automattic. *Essential Security & Performance for WordPress* [online]. 2019 [cit. 2019-05-11]. Dostupné z: `https://jetpack.com/install/personal/get/`.

17. ITHEMES. *BackupBuddy - WordPress Backup Plugin* [online]. 2019 [cit. 2019-05-11]. Dostupné z: `https://ithemes.com/purchase/backupbuddy/`.