TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky a mezioborových inženýrských studií

Studijní program: B 2612 – Elektrotechnika a informatika
Studijní obor: 1802R022 – Informatika a logistika

Webová autentizace proti lokálním a vzdáleným zdrojům

Web site authentication using local and remote resources

Bakalářská práce

Autor: **Vojtěch Kurka**
Vedoucí bakalářské práce: ing. Igor Kopetschke

V Liberci 17. 5. 2007

TECHNICKÁ UNIVERZITA V LIBERCI

Fakulta mechatroniky a mezioborových inženýrských studií

Katedra aplikované informatiky Akademický rok: 2006/2007

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Jméno a příjmení: Vojtěch Kurka

studijní program: B 2612 – Elektrotechnika a informatika

obor: 1802R022 - Informatika a logistika

Vedoucí katedry Vám ve smyslu zákona o vysokých školách č.111/1998 Sb. určuje tuto bakalářskou práci:

Název tématu: **Webová autentizace proti lokálním a vzdáleným zdrojům**

Zásady pro vypracování:

1. Teoretický rozbor problematiky autentizace webových aplikací.
2. Jednotlivé varianty autentizací oproti lokálním a vzdáleným zdrojům.
3. Výhody a nevýhody jednotlivých řešení, doporučení v závislostech na typu projektů
4. Praktická ukázka min. 3 typů autentizací v prostření PHP.

Rozsah grafických prací: dle potřeby dokumentace

Rozsah průvodní zprávy: cca 40 stran

Seznam odborné literatury:

[1] Rosebrock, Eric; Filson, Eric. Linux, Apache, MySQL a PHP : instalace a konfigurace prostředí pro pokročilé webové aplikace . Praha : Grada, 2005.

[2] Bráza, Jiří. PHP 4 : praktické příklady. Praha . Grada, 2003.

[3] Welling, Luke; Thomson, Laura. PHP a MySQL - rozvoj webových aplikací. Praha : SoftPress, 2002.

[4] http://www.php.net

[5] http://www.mysql.com

[6] http://www.root.cz/n/ldap/

[7] http://www.openldap.org

Vedoucí bakalářské práce: ing. Igor Kopetschke

Konzultant:

Zadání bakalářské práce: **22.10.2006**

Termín odevzdání bakalářské práce: **18. 5. 2007**

Technická univerzita v Liberci -12-

.....

Vedoucí katedry

.....

Děkan

V Liberci dne 22.10.2006

Prohlášení

Byl(a) jsem seznámen(a) s tím, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 o právu autorském, zejména § 60 (školní dílo).

Beru na vědomí, že TUL má právo na uzavření licenční smlouvy o užití mé bakalářské práce a prohlašuji, že **s o u h l a s í m** s případným užitím mé bakalářské práce (prodej, zapůjčení apod.).

Jsem si vědom(a) toho, že užít své bakalářské práce či poskytnout licenci k jejímu využití mohu jen se souhlasem TUL, která má právo ode mne požadovat přiměřený příspěvek na úhradu nákladů, vynaložených univerzitou na vytvoření díla (až do jejich skutečné výše).

Bakalářskou práci jsem vypracoval(a) samostatně s použitím uvedené literatury a na základě konzultací s vedoucím bakalářské práce

Datum 17.5. 2007

Podpis

Abstrakt

Tato bakalářská práce popisuje základní postupy při implementaci webové autentizace na LAMP (Linux, Apache. MySQL, PHP) platformě. Zabývá se i autentizací v PHP proti vzdálenému LDAP serveru. Je využito výhradně opensource softwaru, což zaručuje levné nasazení v praxi. Skriptovací jazyk PHP je velmi populární, má širokou podporu v oblasti hostingových služeb a tak řešení podobná těm v této práci není složité použít. Jsou zde i ukázky možností autentizace jen pomocí serveru Apache. Několik kapitol je věnováno návrhům autentizace v typických webových projektech a popisu typů útoků na jejich bezpečnost. Samozřejmě nechybí rady, jak se takovým útokům bránit.

Klíčová slova: autentizace, web, HTTP, LDAP, PHP

Abstract

This bachelor's work concerns basic techniques of the web authentication implementation using LAMP (Linux, Apache. MySQL, PHP) platform. It deals with the authentication using PHP and LDAP server. I have used exclusively opensource software, therefore I expect relatively cheap employment of these methods in practice. The PHP scripting language is highly popular, it is widely supported by web hosting companies. That is why I think these solutions can be easily used in variety of web projects. I present a few standalone solutions using the Apache server. Several chapters describe design of the authentication schemes in typical web projects and attack which are they exposed to. Of course I enclose suggestions how to defend against them.

Keywords: authentication, web, HTTP, LDAP, PHP

Obsah

1 Autentizace uživatele na webu..... 10
2 Obecné přístupy k autentizaci uživatele přes HTTP protokol..... 12
2.1 Charakter HTTP komunikace..... 12
2.2 Cookies..... 15
3 Uživatelská hesla a jejich ověřování a ukládání..... 17
3.1 Vlastnosti hesel..... 17
3.2 Ukládání hesel..... 18
3.3 Ověření hesla..... 18
3.4 Kryptografické hashovací funkce..... 19
4 HTTP server Apache..... 21
4.1 Konfigurační soubory..... 22
4.2 HTTP autentizace..... 23
4.2.1 AuthType Basic..... 23
4.2.2 AuthTypeDigest..... 24
4.2.3 Úložiště přihlašovacích údajů..... 24
5 LDAP..... 26
5.1 Struktura dat..... 26
5.2 LDIF..... 28
5.3 Používáme LDAP při webové autentizaci..... 29
5.4 Komunikace klienta se serverem..... 32
5.5 PHP jako klient pro LDAP..... 32
6 Šifrování přenosu dat..... 35
6.1 Odposlech dat..... 35
6.2 Princip šifrování..... 35
6.2.1 Symetrická šifra..... 35
6.2.2 Asymetrická šifra..... 35
6.3 SSL/TLS..... 36
7 Útoky na bezpečnost aplikace..... 37
7.1 Odposlech komunikace..... 37
7.2 Man in the middle..... 37
7.3 Násilné prolomení hesla..... 37
7.4 SQL injection..... 38
7.5 Cross site scripting..... 38
7.6 Sociální inženýrství..... 39
7.7 Obecné zásady..... 39
8 Příklady metod autentizace podle typu nasazení..... 40
8.1 Osobní webová galerie..... 40
8.2 Internetový obchod..... 40
8.3 Firemní aplikace pro nakládání s citlivými údaji..... 41
9 Praktická ukázka typů webových autentizací..... 42

Slovník znaků, symbolů, zkratek, akronymů, termínů

3DES – Triple Data Encryption Standard

AJAX – Asynchronous Javascript and XML

DAP – Directory Access Protocol

DES – Data Encryption Standard

DSML - Directory Service Markup Language

HTTP – Hypertext Transfer Protocol

IP – Internet Protocol

LDAP – Lightweight Directory Access protocol

LDIF - LDAP Data Interchange Format

OS – Operating System

OSI model – Open Systems Interconnection Basic Reference Model

PHP – PHP Hypertext Preprocessor

PHPDoc – PHP Documentor

SS – Session Stealing

SSL – Secure Socket Layer

TCP – Transmission Control Protocol

TLS – Transport Layer Security

URL - Uniform Resource Locator

XML - Extensible Markup Language

XSS – Cross Site Scripting

Úvod

S potřebou autentizovat uživatele se na webu setkáváme v různých situacích s odlišnými požadavky na implementační náročnost a bezpečnost. Většinou potřebujeme ověřit identitu klienta, abychom mu zobrazili jen ten obsah, který má oprávnění prohlížet, nebo mu nabídli pouze ty funkčnosti, se kterými má právo pracovat. S autentizací na webu úzce souvisí následující 3 pojmy.

Autentizace je proces, kterým ověřujeme, že někdo je opravdu ten, za koho se vydává. Obvykle k tomu používáme kombinaci uživatelského jména a hesla. Existuje ale i mnoho jiných možností, jako čipové karty, snímky oční rohovky, rozpoznávání hlasu nebo otisk prstu.

Autorizace obvykle následuje po autentizaci a slouží k rozhodování, zda autentizovaná osoba má oprávnění provádět požadovanou operaci.

Řízení přístupu může být použito samostatně bez autentizace a určuje, zda uživateli poskytneme určitý obsah nebo funkčnost v závislosti například na aktuálním datu, na IP adrese, ze které se přihlašuje, na prohlížeči, který používá a podobně.

Tyto 3 pojmy se často v oblasti webu velmi prolínají a úzce spolu souvisí, proto se jim budu ve své práci souběžně věnovat. K dispozici máme obvykle mnoho typů autentizace s různými vlastnostmi, výhodami a nevýhodami. Budu se snažit jich zmínit co nejvíce a určit, pro které aplikace jaké co nejvýhodněji použít. K implementaci použiji oblíbenou LAMP platformu (Linux, Apache, MySQL, PHP) a navíc OpenLDAP server.

Mojí snahou je nejen nastínění bezpečných postupů pro úspěšnou autentizaci, ale i ohled na použitelnost autentizace v běžném nasazení. Nesmíme totiž zapomínat, že webové aplikace vyvíjíme pro „běžné uživatele“, kteří nemají jako cíl svojí práce se autentizovat, ale jen rychle a pohodlně použít autentizaci, aby mohli pracovat s aplikací, která je pomocí autentizace zabezpečena. Uživatel je ochoten část svého pohodlí obětovat třeba ve prospěch bezpečnosti svého bankovního účtu, ale musíte ho k tomu hodně přemlouvat. Právě spokojený uživatel je většinou ten, kdo práci programátorů platí.

1 Autentizace uživatele na webu

Při autentizaci uživatele se obvykle zajímáme o tyto věci:

1. Povolit použití webové aplikace tomu, kdo má k vhodné oprávnění
2. Znemožnit použití webové aplikace tomu, kdo na to právo nemá

Tyto dva požadavky se při návrhu aplikace vzájemně ovlivňují. Často platí, že čím více znemožníme průnik nežádoucí osoby, tím větší nároky klademe na oprávněného uživatele při autentizačním procesu. Například musí mít určité vybavení schopné autentizaci provést, může se přihlašovat jen z určitého počítače a podobně. Jakou úroveň zabezpečení nastavit, musíme rozhodnout podle zaměření webové aplikace.

Prvním kritériem pro autentizaci může být **řízení přístupu**. Vstup do aplikace můžeme povolit na základě těchto informací:

- **IP adresa**, ze které se uživatel připojuje. Omezení můžeme nastavit globálně třeba již na firewallu před webovým serverem. To se dělá typicky u interních firemních aplikací. Lze ale i uchovávat IP adresy jednotlivých uživatelů a při změně adresy, ze které se připojuje, ho autentizovat důkladnějším způsobem nebo mu přístup zakázat. Problém je, že více uživatelů se může skrývat za jednou IP adresou při použití překladu IP adres nebo proxy. Také musíme počítat s tím, že některé firmy mají více připojení k internetu, a tak se může IP adresa uživatele v průběhu komunikace měnit
- **Typ webového prohlížeče**, který uživatel používá. Například chceme vynutit používání nejmodernějších prohlížečů, nebo chceme zachytit náhlé změny použitého prohlížeče
- **Vlastnosti dalšího uživatelova vybavení** jako rozlišení obrazovky, verze nainstalovaných pluginů v prohlížeči, podpora cookies. K získání těchto údajů musíme často použít Javascript

Dalším kritériem je obvykle vyžádání přihlašovacích údajů jako **jméno a heslo**. Lze ale také použít třeba **klientský certifikát**. V mé práci se zabývám pouze kombinací jména a hesla, protože nepředpokládám, že by někdo používal na platformě PHP k

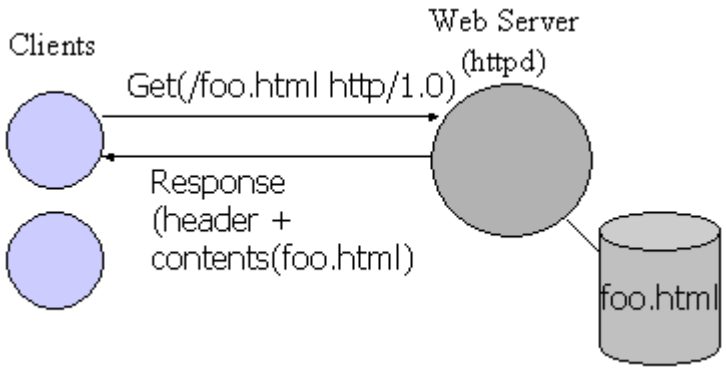
autentizaci klientské certifikáty.

V průběhu autentizace potřebujeme zajistit, aby nějaká třetí osoba nemohla posílané autentizační údaje zjistit. Toho obvykle dosáhneme použitím **šifrovaného HTTP přenosu**. Je třeba také myslet na to, že ke zjištění přihlašovacích údajů může útočník použít některou z násilných metod, jako zkoušení **náhodných hesel**, nebo **slovníkový útok.**

2 Obecné přístupy k autentizaci uživatele přes HTTP protokol

2.1 Charakter HTTP komunikace

Uživatel využívá webovou aplikaci tím, že zasílá jednotlivé HTTP požadavky a pomocí nich mu server zasílá požadovaná data, nebo provádí operace vyžádané těmito HTTP požadavky. Problém je v tom, že protokol HTTP je bezstavový. Neudržuje si žádnou informaci o souvislosti po sobě jdoucích požadavků. To znamená, že pokud v prvním požadavku autentizujeme uživatele, v druhém už nevíme, zda se jedná opravdu o stejného uživatele, který se před chvílí úspěšně autentizoval. Požadovaný stav musíme implementovat sami do naší webové aplikace, nebo autentizovat každý požadavek zvlášť.



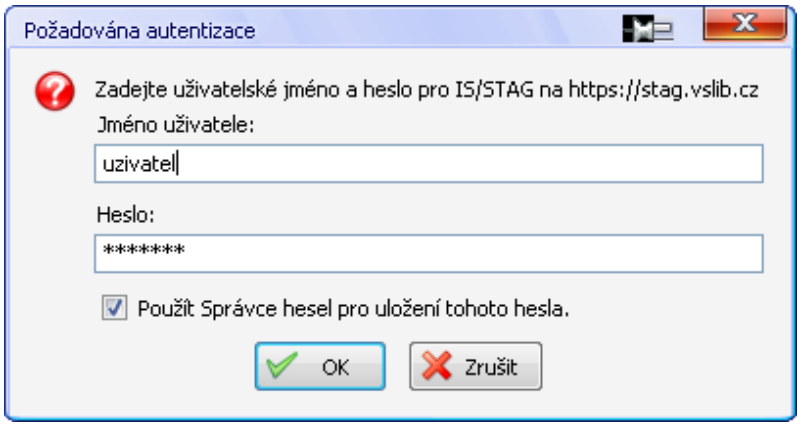
Obr. 1: HTTP komunikace klienta s webovým serverem

Dva základní přístupy jsou:

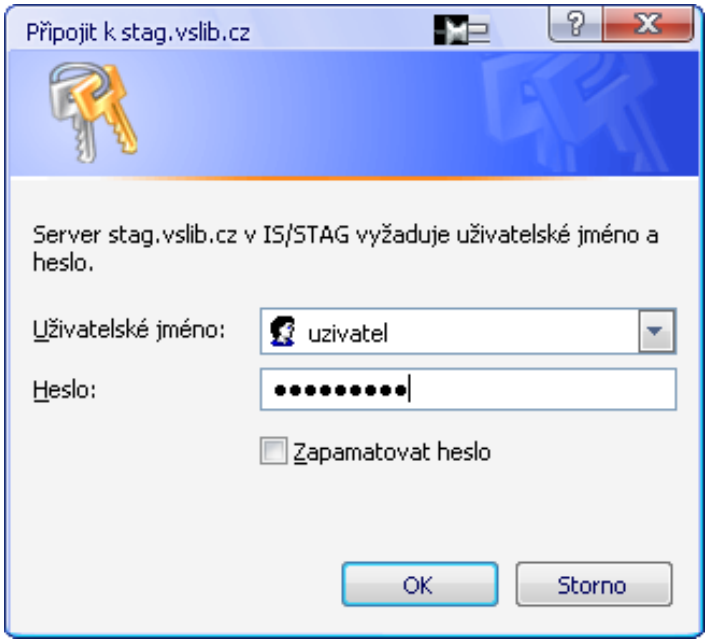
1. Klient zasílá přihlašovací údaje s každým HTTP požadavkem, server je pokaždé zkontroluje a podle toho uživatele autentizuje
2. Klient zašle přihlašovací údaje s prvním požadavkem. Server mu přidělí jeho unikátní identifikátor, který pak klient zasílá serveru s každým dalším požadavkem místo přihlašovacích údajů

První řešení je již obsažené v HTTP protokolu. Server pošle klientovi HTTP hlavičky *„401 Authentication Required“* a „*WWW-Authenticate: Basic realm="realm"*“. „*Realm*“ je název této autentizace a prohlížeč ho použije k uložení

hesla, stejné heslo pak posílá tomuto serveru, jakmile je požadována autentizace se stejným názvem realm. Prohlížeč na tyto dvě hlavičky zareaguje tím, že uživateli zobrazí přihlašovací box s políčky pro přihlášení (pokud přihlašovací údaje už nebyly zadány). Uživatel zadá přihlašovací údaje a ty se odešlou na server k ověření.



Obr. 2: Mozilla Firefox, HTTP autentizace



Obr.3: Internet Explorer 7, HTTP autentizace

Klientský prohlížeč si zapamatuje tyto údaje, takže není nutné je zadávat s každým požadavkem znovu. Na serveru můžeme využít k autentizaci přímo funkčnost webového serveru, nebo s přihlašovacími údaji pracujeme třeba v PHP (proměnné *$_SERVER['PHP_AUTH_USER']* a *$_SERVER['PHP_AUTH_PW']*).

Nevýhody tohoto způsobu:

1. Přihlašovací údaje jsou posílány s každým požadavkem, což je velice riskantní, pokud nepoužíváme šifrování přenosu
2. Prohlížeč si pamatuje přihlašovací údaje dokud ho neukončíme, proto se uživatel nemůže z webové aplikace odhlásit jiným způsobem, než že zavře okno prohlížeče. V prohlížeči Firefox 2.0 můžeme použít funkci „Vymazat důvěrná data=>Relace s autentizací.“, v prohlížeči Opera 9.0 nalezneme obdobnou funkci, v Internet Exploreru 7 ale nepochodíme
3. Nemáme možnost změnit formát dat odesílaných z prohlížeče na server, vždy to bude prostý text, velice jednoduchý na odposlechnutí útočníkem, heslo není nijak chráněno proti přečtení
4. Vzhled – nemůžeme nadefinovat umístění a vzhled přihlašovacího okna v prohlížeči, to je dáno pouze tím, jak ho implementoval výrobce webového prohlížeče

Druhé řešení implementujeme tak, že po úspěšné autentizaci uživatele mu přidělíme jedinečný identifikátor, který nám zasílá s dalšími požadavky. Potřebujeme ale tento identifikátor někde u uživatele (na serveru samozřejmě také) uložit pro jeho zachování mezi dvěma požadavky. Obecně se využívá dvou možností:

- Identifikátor pošleme klientovi pokaždé ve stránce tak, že na konci každého odkazu přidáme identifikátor jako proměnnou v GET požadavku, nebo jako skrytou položku formuláře pro zaslání POST pořadavkem.
- Identifikátor pošleme klientovi jednorázově jako cookie, kterou si uloží do prohlížeče

Osobně preferuji zásadně použití cookies a také ho všem doporučuji. Předávání v URL mi připadá neohrabané, má mnoho nevýhod:

- Identifikátor je přímo ve stránce a odkazu. Případný útočník ho může získat pomocí reference na poslední navštívenou stránku (tzv. *referer*) nebo z uloženého, případně omylem zaslaného odkazu nějaké třetí osobě. Běžný uživatel netuší, jakou funkci tento identifikátor má, proto musíme předpokládat že s ním bude nakládat po svém a bez omezení
- Identifikátor je na mnoha místech webové stránky, proto se stránka těžko dá

někde uložit do cache. Respektive, uložit jde, ale podruhé se pro jiného uživatele nedá použít, protože má uživatelsky specifický obsah

- Pokud použijete bezpečnější variantu, u které generujete identifikátor znovu při každém požadavku, nelze použít AJAX nebo ochranu jiných dat načítaných zvlášť do stránky, jako jsou obrázky
- Poslední vada je kosmetická – URL s dlouhým identifikátorem nevypadá nejlépe

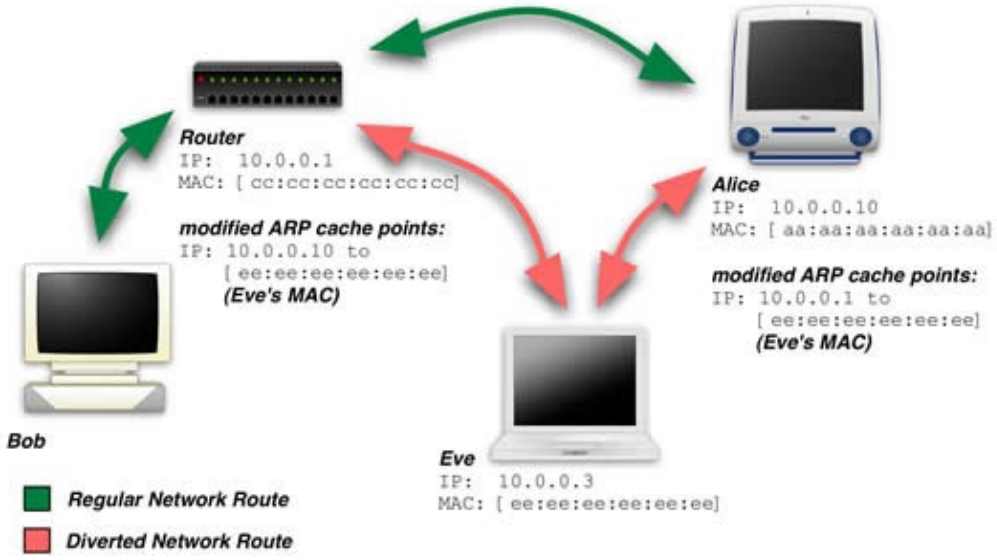
2.2 Cookies

Cookies jsou proměnné, které nám obvykle webový prohlížeč dovolí uložit u uživatele. Obsahují dva bezpečnostní prvky:

1. Čas, do kdy je má prohlížeč uchovat. Pokud čas neurčíme, má cookie platnost do uzavření prohlížeče
2. Název webového serveru, který s obsahem cookie smí pracovat. To znamená, že jakmile do prohlížeče načteme třeba *seznam.cz*, tak tato stránka nesmí otevřít cookie, která byla nastavena pro server *centrum.cz*

Trochu problém může být fakt, že někteří uživatelé použití cookies ve svém prohlížeči vypínají. Nemyslím ale, že by to byl velký podíl, proto **doporučuji používat výhradně cookies** a uživatele, kteří cookies nepoužívají, k jejich použití donutit. Možná se setkáte s tím, že zákazník prohlásí, že mu nefunguje přihlášení a ať to koukáme spravit. Situace je obzvlášť veselá, když je to zrovna ředitel firmy, pro kterou pracujete na velké zakázce. V takovém případě doporučuji maximalizovat vaše úsilí v polopatickém vysvětlení důležitosti vašeho řešení pro bezpečnost dat, se kterými bude aplikace pracovat.

Pro maximalizaci bezpečnosti můžete identifikátor uživatele **generovat při každém požadavku** nový, tím se výrazně zkrátí doba, po kterou může útočník ukradený identifikátor použít. Samozřejmě nám to nebude nic platné, pokud útočník komunikaci odposlouchává, tomu zabráníme jedině šifrováním přenosu dat. Jako malá berlička může sloužit kontrola náhlé změny specifických údajů o uživateli, jako IP adresa nebo verze prohlížeče. To ale nepomůže v případě, že je útočník schopen data po cestě nejen odposlouchávat, ale i měnit (tzv. „Man in the middle“).



Obr.4: útok "Man in the middle", trasa komunikace převedena přes útočníka

Bezstavovost HTTP protokolu nám přináší ještě jedno úskalí, nevíme, kdy uživatel přestal s webovou aplikací pracovat. Získaný bezpečnostní identifikátor tak vlastně platí stále, ale my netušíme, že uživatel již dávno práci ukončil. Můžeme mu sice umožnit provést odhlášení pomocí nějaké akce na stránce (stisk tlačítka, klepnutí na odkaz), ale musíme počítat s variantou, kdy to neprovede. Lze to ošetřit Javascriptovou funkcí, která odhlášení provede sama ve chvíli opuštění stránky s aplikací. Na to se ale také nemůžeme spolehnout, obecně používané řešení proto je nastavení **časově omezené platnosti identifikátoru**. Například jakmile uživatel nepošle další HTTP požadavek během posledních 3 minut, identifikátor na serveru ztrácí platnost a uživatel je automaticky odhlášen.

3 Uživatelská hesla a jejich ověřování a ukládání

3.1 Vlastnosti hesel

Nároky na uživatelské heslo na webu:

- **zapamatovatelné** – uživatel často používá více počítačů a nechce hesla nosit s sebou někde uložená
- **dostatečně odolné** proti násilným útokům

Opět zde řešíme dvě protichůdné věci a je třeba vybrat nějaký kompromis. Heslo by mělo být odolné proti **násilnému prolomení**, to znamená, že chceme útočníkovi znemožnit jeho nalezení pomocí zkoušení náhodných hesel. K tomu na webu stačí heslo dlouhé cca 7-8 znaků, protože je nepravděpodobné, že by útočník stihl vyzkoušet všechny varianty 7 znaků v nějakém reálném čase. Na webu to útočníkovi ještě ztěžuje latence síťové konektivity a reakce serveru, proto by pro takový útok mohla být dostatečná překážka i délka hesla třeba jen 6 znaků. Osobně zastávám názor, že bychom měli uživatele nutit zadávat hesla o minimální délce 8 znaků, aby si nezvykli krátká hesla používat i v desktopových aplikacích, kde lze násilný útok provádět řádově rychleji.

Útočník může jednak zkoušet náhodná hesla, ale hlavně může využít tzv. **slovníkový útok**. Ten spočívá v tom, že testuje jen slova obsažená v nějakém slovníku, tím získá útočník užší výběr s větší pravděpodobností nalezení hesla. Takový útok bývá velice efektivní hlavně v prostředí neškolených uživatelů, kteří používají běžná slova a jména jako hesla.

Řešení kvality hesla se dá vyřešit tím způsobem, že uživateli heslo vygenerujeme a nenecháme ho použít vlastní. Za to vás ale většina uživatelů nebude mít ráda a navíc vám přibude mnoho emailů s prosbou o obnovení zapomenutého hesla.

Doporučuji tedy nechat volbu hesla na uživatelích a nutit je, aby heslo bylo dlouhé alespoň 8 znaků a obsahovalo jak písmena, tak číslice. Lze i použít slovník pro ověření hesla, to ale považuji u hesel složených z číslic a písmen za zbytečné, použil bych to jen u hesel, která se skládají pouze z písmen.

Nastává otázka, jaké znaky v heslech povolit. Zde dávám přednost tomu, aby

byla malá pravděpodobnost, že se běžný uživatel splete. To znamená, používat jen malá písmena a čísla. Bez mezer, bez diakritiky, bez rozlišení velikosti písmen, popřípadě i bez speciálních znaků. Myslím, že ve většině webových aplikací je důležitější fakt, že se se oprávněný uživatel do aplikace dostane a nenabude dojmu, že heslo zapomněl. Za užitečnou věc považuji v Čechách i nahrazení „Y“ za „Z“. Prakticky to znamená, že tyto dva znaky v ověřovacím procesu považujeme za totožné. Tím se vyhneme omylům uživatelů při nechtěném přepnutí módu klávesnice, kdy uživatel heslo při psaní nevidí, a tak se nemůže o této chybě dozvědět. Funkce pro uložení/ověření hesla by mohla vypadat takto:

1. odstraň nepovolené znaky
2. převeď na malá písmena
3. převeď znaky s diakritikou na znaky bez diakritiky
4. nahraď „Y“ za „Z“
5. porovnej hesla

Stále ale platí zásada, že chci sice uživateli jeho přihlášení maximálně ulehčit, ale ne na úkor bezpečnosti. Veškeré změny oproti standardním postupům byste proto měli dělat s velkým rozmyslem.

3.2 Ukládání hesel

Asi je jasné, že uživatelská hesla bychom měli ukládat na bezpečném místě. I přes všechna zabezpečení musíme předpokládat nejhorší případ, kdy by někdo mohl úložiště s hesly uživatelů odcizit. Základní zásada je neukládat hesla v prostém textu (plaintext). Bezpečný způsob je při ukládání vygenerovat otisk hesla pomocí jednocestné kryptografické hashovací funkce. Do úložiště (databáze, textový soubor...) pak ukládáme tento otisk a ne heslo, při ověření porovnáme tyto dva otisky, ten uložený a druhý vygenerovaný z aktuálně zadaného hesla.

3.3 Ověření hesla

Proces ověření může mít mnoho podob:

- Klient pošle heslo v čistém textu na server, tam z něj uděláme hash a

porovnáme s hashem uloženým v úložišti. Toto řešení vyžaduje šifrování přenosu, jinak by útočník mohl hesla jednoduše odposlechnout

- Klient odešle na server hash hesla a my ho tam porovnáme s hashem v úložišti. Útočník může hash odposlechnout a lehce použít pro svoji autentizaci, ale alespoň nebude znát tvar hesla, a nemůže ho použít v jiných webových aplikacích, kde uživatel používá shodné heslo. Často se totiž stává, že útočník odposlechne heslo u přihlášení k emailu nebo nějaké triviální webové službě, a poté zkusí to samé heslo použít k autentizaci u internetového bankovnictví stejného uživatele. Pravděpodobnost, že se trefí je bohužel poměrně velká, protože povědomí běžných uživatelů o tomto riziku je malé
- Pošleme klientovi náhodné číslo, klient udělá hash z tohoto čísla společně s heslem a celý ho pošle na server, kde uděláme stejnou operaci s heslem uloženým v databázi a číslem, které jsme klientovi zaslali. Toto řešení zabezpečí heslo proti jeho odposlechnutí, protože přenášený hash platí pouze pro první autentizaci s tím to hashem na serveru, ale není odolné proti útoku „Man in the middle“, kdy útočník sedí někde mezi klientem a serverem, vydává se při přenosu za obě strany a pozměňuje posílaná data

U práce se zadanými přihlašovacími údaji v aplikaci platí to samé, jako u kterýchkoliv jiných dat zasílaných uživatelem. Musíme počítat s tím, že na server může přijít cokoliv. Je třeba data ošetřit proti vkládání nebezpečných kódů, které by mohly změnit chování aplikace, způsobit SQL injection a jiné nežádoucí akce.

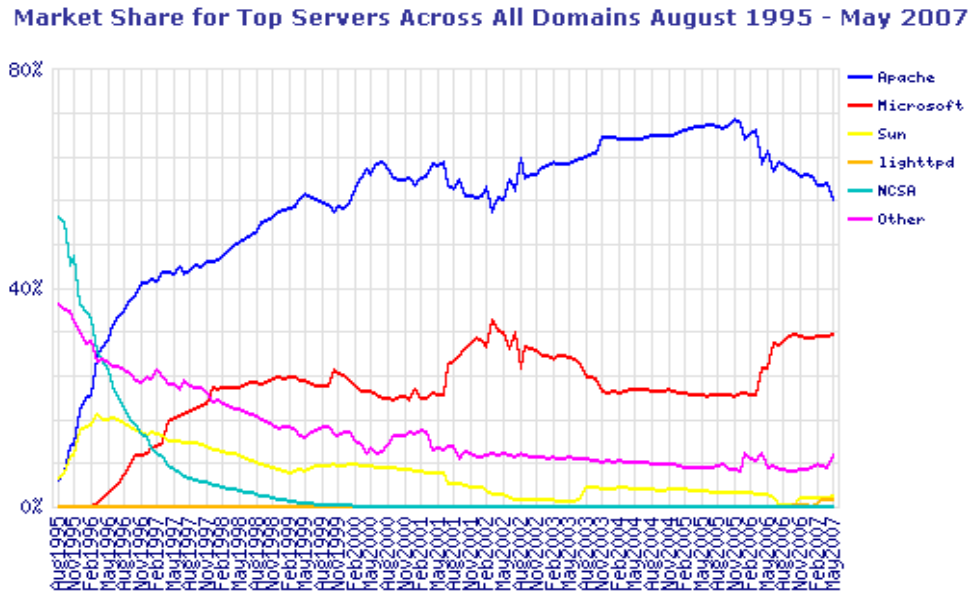
3.4 Kryptografické hashovací funkce

Jak bylo výše naznačeno, hashovací funkce mají při autentizaci široké použití. Hashovacích funkcí ale existuje spousta typů a ne všechny jsou vhodné pro autentizační účely. Při autentizaci nás zajímají hlavně jednocestné kryptografické hashovací funkce. Pomocí takových získáme otisk určitých dat, ale nemůžeme získat původní podobu těch dat na základně samotného otisku. Po takových funkcí požadujeme, aby byly bezpečné, to znamená, aby nikdo na základě znalosti otisku dat nemohl najít jiný vzorek dat jemu odpovídající. Pokud v aplikaci přenášíme otisky hesel, potřebujeme, aby na základě těchto otisků někdo nevygeneroval (v reálném čase) jiné heslo, které nám poté podstrčí.

Tyto hashovací funkce mají typickou vlastnost, stárnou. Se zvyšováním dostupného výpočetního výkonu a objevování nových metod pro jejich prolomení se snižuje jejich bezpečnost. Proto je nutné sledovat vývoj v této oblasti a v aplikacích používat jen takové, které jsou ještě považovány za bezpečné. Přibližně před rokem proběhla internetem zpráva, že skupina čínských vědců dokáže poměrně rychle nacházet kolize k funkci **MD5**. Se vzrůstajícím počtem jejích prolomení se stává pro bezpečnostní účely nepoužitelnou. Dokonce i pokročilejší alternativa, **SHA-1** se otřásá v základech [Plíva V., 2006]. Zatím si musíme vystačit s hashovací funkcí SHA-1, dokud nebudou dostatečně implementovány pokročilejší alternativy. Funkce MD5 již byla již dostatečně prolomena na to, abychom ji mohli považovat za nedostatečně bezpečnou.

4 HTTP server Apache

Apache je v současnosti nejpoužívanější HTTP server. Poslední průzkumy hovoří o přibližně **60% webových serverů** [Netcraft.com, 2007].

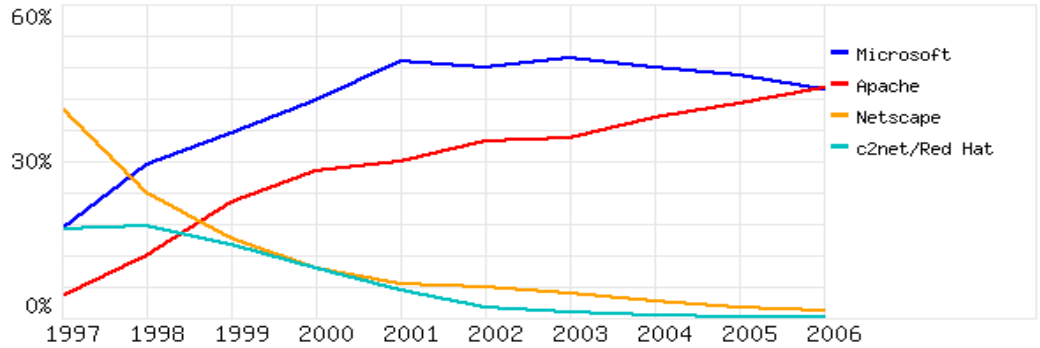


Top Developers

Developer	April 2007	Percent	May 2007	Percent	Change
Apache	66899485	58.86	66087698	56.00	-2.86
Microsoft	35380121	31.13	37170290	31.49	0.36
Sun	1907610	1.68	2141252	1.81	0.13
lighttpd	1382843	1.22	1411788	1.20	-0.02
Zeus	488838	0.43	491989	0.42	-0.01

Obr.5: Statistika používání serveru Apache

Také na poli využití SSL/TLS pro šifrovanou HTTP komunikaci má silné postavení [Netcraft.com, 2006].



Obr.6: Statistika používání serveru Apache pro SSL komunikaci

K radosti všech uživatelů je opensource a má velmi volnou licenci, která nenutí ke zveřejňování zdrojových kódů, při použití Apache jen musíte zachovat jméno původního softwaru.

Apache nám už sám o sobě nabízí moduly pro HTTP autentizaci, jednoduše nastavitelné pomocí konfiguračních souborů.

4.1 Konfigurační soubory

Apache nabízí pro konfiguraci více možností. Základní je soubor *httpd.conf*, určující hlavní konfiguraci. V něm lze určit globální nastavení serveru, nebo využít sekcí <*Directory*> pro podrobnější nastavení zvlášť pro určité adresáře. Pro uplatnění změn je nutné server restartovat, aby načetl novou konfiguraci. Před použitím nového konfiguračního souboru je dobré zkontrolovat jeho syntaxi, jinak server odmítne naběhnout:

```
/etc/init.d/httpd configtest
```

Pro načtení nové konfigurace nemusíme Apache restartovat klasickým způsobem, ale stačí použít tzv. g*raceful* restart:

```
# klasický restart
/etc/init.d/httpd restart
# lehký restart
/etc/init.d/httpd graceful
```

Graceful pouze nařídí potomkům hlavního procesu, aby se ukončili ihned po skončení vyřizování aktuálního požadavku a hlavní proces mezitím načte novou konfiguraci. Po skončení běhu starých potomků jsou tito ihned nahrazeni novými s již novou konfigurací a ihned obsluhují nové požadavky klientů.

Další možností jsou soubory *.htaccess*, které lze umísťovat libovolně do adresářové struktury a které určují nastavení pro daný adresář. Tyto soubory jsou parsovány pokaždé, když klient zašle požadavek na soubor z tohoto adresáře, tudíž není nutné server restartovat pro načtení nové konfigurace. Aby Apache použil direktivy,

které do tohoto souboru umístíte, je nutné jejich použití povolit v hlavním konfiguračním souboru *httpd.conf* :

```
<Directory "/var/www/html">
    AllowOverride All
</Directory>
```

Většinou je vhodnější specifikovat jen určité skupiny direktiv, které chcete povolit použít:

```
<Directory "/var/www/html">
    AllowOverride AuthConfig
</Directory>
```

Direktiva *AllowOverride* smí být použita jen v sekcích <*Directory*>

4.2 HTTP autentizace

Při požadavku klienta Apache pošle zpět HTTP hlavičku *„401 Authentication Required "* a klientský prohlížeč uživateli nabídne okno pro zadání uživatelského jména a hesla. Apache poté vyhodnotí správnost zaslaných přihlašovacích údajů a při kladném výsledku pošle klientovi požadovaný obsah.

Apache nabízí dva typy autentizace - *AuthType Basic* a *AuthType Digest*. Liší se ve formátu hesel při přenosu. *Basic* posílá hesla zakódovaná v *base64*, to znamená jednoduše čitelná pro útočníka. *Digest* využívá MD5 hash přihlašovacích údajů společně s dalšími údaji.

4.2.1 AuthType Basic

Varianta *AuthType Basic* byla definována již ve verzi protokolu HTTP 1.0. Snad všechny známé moderní prohlížeče ji podporují. Ke spuštění autentizace na určitém adresáři se používá většinou souborů *.htaccess*, stačí do něj napsat:

```
AuthType Basic
AuthName "Test AuthBase"
AuthUserFile /var/www/html/.htpasswd
Require user vojta
```

Direktivou *Require user* omezujeme vstup pouze na určeného uživatele. Lze ale

také nastavit povolení pro jakéhokoliv platného uživatele nebo na skupinu. Direktivou *AuthUserFile* určujeme cest k souboru s hesly, který Apache použije pro ověření. Samozřejmě ho musíme nejdřív vygenerovat, pro prvního uživatele použijeme:

```
htpasswd -c /var/www/html/ vojta
```

Pro další uživatele už jen:

```
htpasswd /var/www/html/ vojta2
```

Implicitně se hesla na platformě Linux ukládají zakódované pomocí systémové funkce *crypt*. Můžeme použít ale i ukládání pomocí MD5 hashe:

```
htpasswd -m /var/www/html/ vojta2
```

Popřípadě pomocí SHA hashe:

```
htpasswd -s /var/www/html/ vojta2
```

Metody uložení hesla lze i kombinovat v rámci jednoho souboru. Soubory s hesly byste měli uložit odděleně od stromu webu a nastavit jim práva pouze pro čtení Apachem.

4.2.2 AuthTypeDigest

Varianta *Digest* byla definována až ve verzi protokolu HTTP 1.1. Je bezpečnější, protože přihlašovací údaje kóduje pomocí MD5 hashe dohromady s náhodnými daty zaslanými ze serveru a URL požadavku. Bohužel zatím ještě není bezchybně podporována prohlížeči, konkrétně Internet Explorer 6 má problém při použití s GET požadavky. [Apache.org, 2004, manuál k verzi 2.2]. „Řešením“ je používat výhradně POST požadavky. Od verze Apache 2.0.51 je k dispozici hack:

```
BrowserMatch "MSIE" AuthDigestEnableQueryStringHack=On
```

4.2.3 Úložiště přihlašovacích údajů

Ukládání přihlašovacích údajů se řídí direktivou *AuthBasicProvider,* popřípadě. *AuthDigestProvider*. Výše jsem ukázal využití textového souboru. Další možnosti, které Apache nabízí jsou:

- **databázové soubory** místo textových. Jsou na bázi *Berkeley DB* (modul *mod_auth_db*). Efektivnější pro větší počty uživatelů než několik set, kdy by vyhledání v neindexovaném textovém souboru trvalo déle

- **vzdálený LDAP server** (modul *mod_auth_ldap)* – lze využít i TLS nebo SSL pro šifrování přenosu

Další podrobné popisování použití způsobu autentizace pomocí Apache mi připadá jako „nošení dříví do lesa“, protože máme k dispozici velmi kvalitní a obsáhlý online manuál na webu Apache serveru.

5 LDAP

LDAP je Lightweight Directory Access Protocol tzn. standardizovaný protokol pro přístup do datových adresářů. Vychází ze staršího protokolu X.500, který je komplexnější, složitější a pro většinu aplikací zbytečně komplikovaný. V dnešní době protokol X.500 využívají jen některé velké telekomunikační firmy, jinak se používá LDAP, Microsoft Active Directory a podobné produkty.

LDAP je přenášen přímo na běžném TCP/IP protokolu, nevyžaduje OSI síťový model jako X.500.

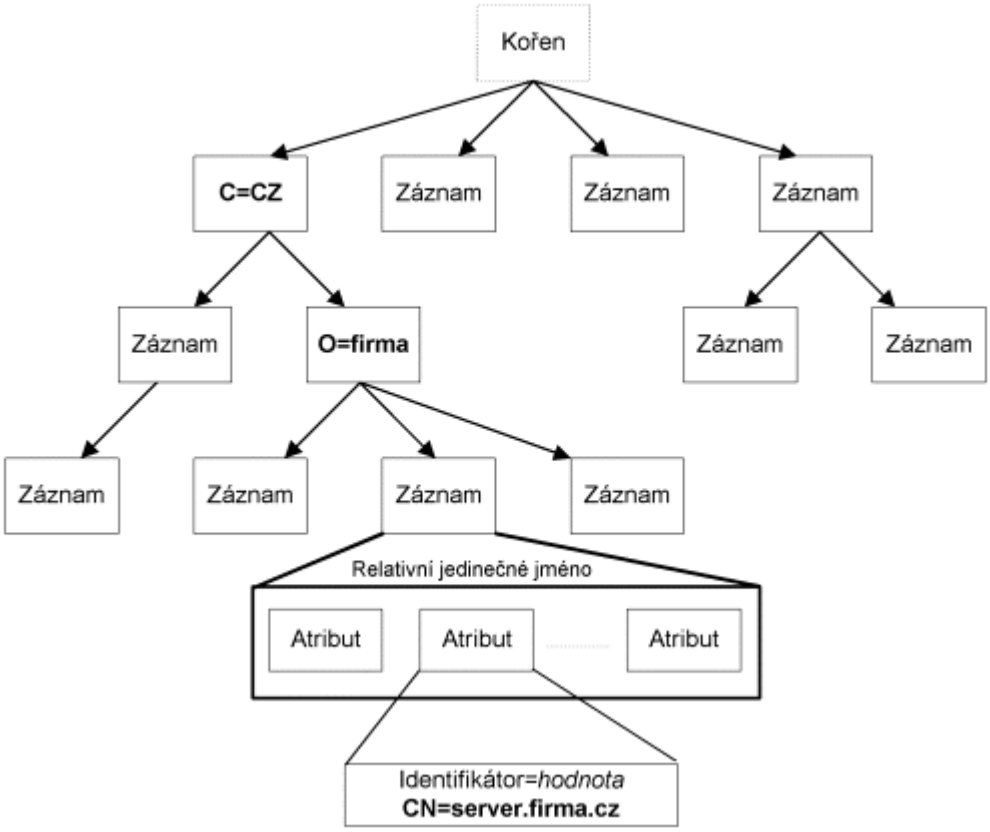
Adresářová služba LDAP jako celek je:

- protokol LDAP
- serverový software pro poskytování dat
- klientský software
- LDIF – standardizovaný formát pro výměnu adresářových dat

5.1 Struktura dat

U LDAPu se často nabízí srovnání s relačními databázemi. LDAP ale jde jinou cestou. Data uskladňuje ve stromové struktuře pomocí schémat. Nepoužívá relace ani transakce, neobsahuje integritní omezení, pokud samotné schéma za integritní omezení nepovažujeme. Jeho primární určení je ukládat data tak, aby se v nich dalo rychle vyhledávat a málokdy je modifikovat. Stromová struktura je výhodná, protože odráží naše okolí, lehce lze zachytit souborový systém, DNS, strukturu oddělení ve firmě, rozdělení zaměstnanců atd.

Strom vždy začíná u nějakého kořene a dále se větví.



Obr.7: Struktura dat na LDAP serveru

Každá položka může mít libovolné množství atributů různých typů. To, jaké atributy položky mají, určujeme pomocí schémat. Schéma je soubor pravidel, který určuje vlastnosti, počet a význam atributů u jednotlivých položek. Definujeme v něm tzv. objektové třídy (object classes) a ty určují, co do adresáře smíme uložit. LDAP vždy kontroluje vkládané hodnoty, aby se shodovaly s nastaveným schématem. Třídy mohou od sebe navzájem dědit vlastnosti podobně jako v objektových programovacích jazycích. Rozdíl je ten, že nedefinujeme žádné metody pro práci s objekty, jen jejich vlastnosti. U každé položky je povinný alespoň jeden atribut *objectClass*, který právě určuje třídu použitou pro tuto položku. Základní třídou je třída *top* a od ní dědí všechny ostatní třídy. Schéma můžete použít buď standardizované a získáte tak výhodu při spolupráci mezi více LDAP servery, nebo si nadefinujete své vlastní, přesně vyhovující vašim potřebám.

Každá položka ve stromu má jednoznačné jméno, podle kterého ji můžeme s jistotou vždy najít. Říkáme tomu *distinguished name* (zkráceně se používá DN) a tvoří

se hierarchicky z posloupnosti DN nadřazených prvků. S tím souvisí i pojem *relative distinguished name*, kterým označujeme rozlišující název (dvojice <atribut>=<hodnota>) položky na aktuální hladině stromu. Pokud k DN nadřazené položky přidáme libovolný RDN aktuální položky, získáme tak DN té aktuální. Na serveru ještě definujeme suffix, což je jméno nejvýše umístěné položky ve stromu, například „o=mojefirma,c=cz“ pro firmu Mojefirma z ČR.

5.2 LDIF

LDIF je standardizovaný textový formát pro výměnu dat s LDAP serverem. Všechna data uložená na serveru lze převést do formátu LDIF, binární data se do něj kódují pomocí *base64*. Je to obdoba SQL dumpu u databází, jen potřebujete ještě schéma pro zrekonstruování datového stromu. Kromě LDIFu se prosazuje i novější formát DSMLv2 založený na XML.

Příklad LDIF souboru:

```
# první záznam
dn: cn=Pepek P Namornik,dc=firma,dc=cz
cn: Pepek J Namornik
cn: Pepek Namornik
objectClass: person
sn: Namornik

# druhý záznam
dn: cn=Pepka P Namornikova,dc=firma,dc=cz
cn:  Pepka P Namornikova
cn:  Pepka Namornikova
objectClass: person
sn: Namornikova
# Base64 encoded JPEG photo
jpegPhoto:: /9j/4AAQSkZJRgABAAAAAQABAAD/2wBDABALD
 A4MChAODQ4SERATGCgaGBYWGDEjJR0oOjM9PDkzODdASFxOQ
 ERXRTc4UG1RV19iZ2hnPk1xeXBkeFxlZ2P/2wBDARESEhgVG
```

Jednotlivé položky se oddělují prázdným řádkem, popis položky se skládá z

dvojic <atribut>: <hodnota>. U názvů atributů se obvykle využívají ustálené aliasy jako SN (Surname, příjmení), CN (Common name, jméno). Pokud je hodnota atributu delší než jeden řádek, zajistíme to přidáním jedné mezery na začátku každého následujícího řádku.

LDIF lze výhodně použít k offline práci s daty na LDAP serveru jako je import a export dat z jiného serveru nebo aplikace. Existuje množství programů a skriptů, pomocí kterých například databázová nebo XML data jednoduše převedete do LDIFu a pak rovnou jednorázově nahrajete do LDAP serveru. Například uživatelské účty na serveru takto lze do LDAPu převést za pár minut.

Při samotné síťové online komunikaci se serverem se ale LDIF nepoužívá, místo toho má LDAP sadu pravidel LBER (Lightweight Basic Encoding Rules), která určují kódování dat při přenosu. Je to datově úspornější metoda pro síťový přenos než přenášet textová data LDIFu.

5.3 Používáme LDAP při webové autentizaci

Nastává otázka, jaké postavení má LDAP při webové autentizaci, kdy ho použít, kdy použít něco jiného atd. Primární určení LDAPu je univerzální strukturované úložiště dat s rychlým přístupem. Jaké jsou výhody a nevýhody z hlediska webového použití?

Výhody:

- **rychlý přístup**. LDAP je stvořený pro vyhledávání a čtení dat, maximálně urychlené indexací položek
- přirozená stromová struktura – často odráží realitu, kterou chceme pomocí datového úložiště zachytit
- **standardizovanost** – zajišťuje velké rozšíření a rozmanitost použití
- **množství implementací** – jak na serverové, tak na klientské straně existuje mnoho hotových řešení, často opensource. Velké spektrum aplikací je na spolupráci s LDAPem připraveno, například webový server Apache, emailoví klienti, přihlášení do OS
- **zajištění dostupnosti služeb** – pomocí replikace lze zvýšit dostupnost a

zatižitelnost služby

- **zabezpečení** – lze flexibilně nastavovat oprávnění uživatelů pro práci s adresářem
- rozmanitost ukládaných dat – lze ukládat vše od textu přes obrázky až po digitální certifikáty

Nevýhody:

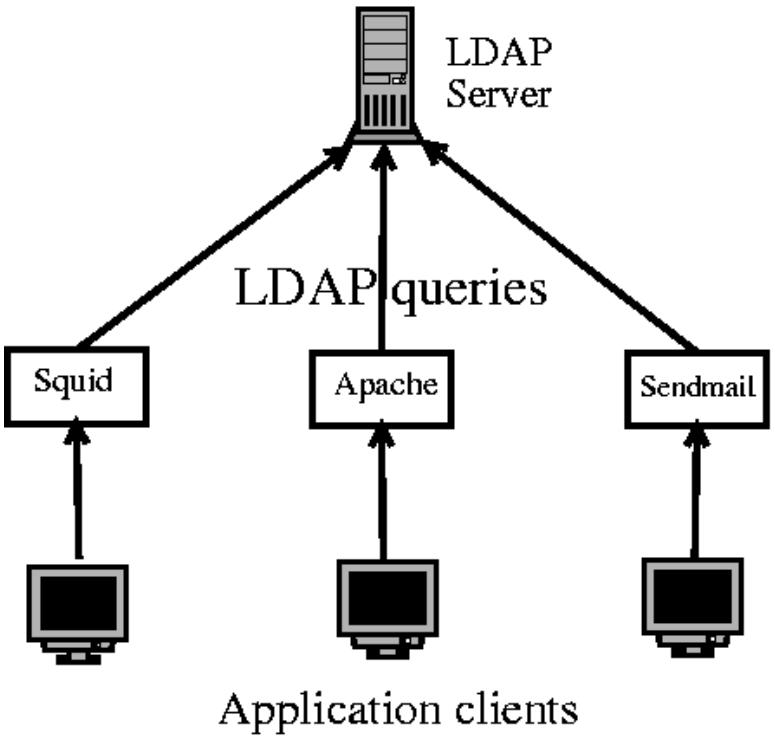
- **pomalý zápis** – jakékoliv změny dat jsou pomalejší než u databázových systémů
- chybějící integritní omezení a referenční integrita – tyto vlastnosti nabízí v omezené míře jen některé komerční produkty
- **absence transakcí** – obvykle u serverových implementací nejsou k dispozici
- omezené možnosti dotazování – LDAP nenabízí pokročilé komplikované nástroje pro dotazování, agregační a manipulační operace jako jazyk SQL. Je postaven na jednoduchém principu „znám hledaný objekt, chci zjistit nějakou jeho vlastnost“
- **omezený výběr opensource serverových řešení** – oproti databázovým systémům neexistuje tolik vyzrálých opensource řešení jako na poli databázových serverů
- **existuje méně kvalifikovaných lidí pro práci s LDAPem** – v oblasti webových aplikací rozumí mnohem více programátorů a administrátorů databázím než LDAPu, od toho se odvíjí i tržní cena kvalifikovaných lidí

V mnoha případech nejsou ani tak důležité výhody a nevýhody jako fakt, že máme hotový informační systém fungující na adresářové službě a potřebujeme ho napojit na webovou aplikaci. V takovém případě je zbytečné data replikovat do databáze, často to firma i striktně zakazuje kvůli bezpečnosti. Potom stačí využít jako úložiště dat jen LDAP, popřípadě doplnit o databázový systém pro data, která souvisejí čistě s webovou aplikací.

LDAP je užitečný hlavně v případě, kdy potřebujeme, aby úložiště dat sloužilo nejen webové aplikaci, ale i k množství jiných operací. Ve firmě například potřebujeme

jednotný adresář uživatelů, do kterého přistupují:

- operační systémy pro ověření uživatelů na stanicích
- mailservery pro ověření přístupu uživatelů ke schránkám
- emailoví klienti pro získání emailových adres kolegů z oddělení
- karetní čtečky pro přístup do místností, skladů a jiných úložných prostor
- programy pro práci využívající přihlášení uživatelů
- programy ukládající uživatelovo nastavení do centrálního úložiště



Obr.8: Využití LDAP serveru

V takovém heterogenním prostředí je LDAP dobrou volbou, protože nám svým standardizovaným protokolem umožní jednoduchou součinnost různorodých služeb a administrátoři lehce nastaví oprávnění uživatelů ke všem službám přehledně na jednom místě.

Pokud vyvíjíme webovou aplikaci, která má vlastní databázi a nekomunikuje s okolím jinak než webovými službami, nemá moc smysl kvůli autentizaci uživatelů nebo uchovávání jejich dat zřizovat LDAP server. Databázový server potřebujeme tak jako

tak a LDAP server by byl další místo vzniku možných problémů a nekompatibilit. Může se hodit jen ve specifických případech, kdy například potřebujeme objektovou stromovou strukturu místo relační databáze, popřípadě odlišnou politiku uživatelských oprávnění pro přístup k datům.

5.4 Komunikace klienta se serverem

Typicky probíhá komunikace takto:

1. Klient otevře TCP spojení (popřípadě použije i SSL/TLS) s LDAP serverem na TCP portu 389 a zašle příkaz *bind*, který slouží pro autentizaci klienta. Pokud jde o anonymní spojení, klient neodešle žádné přihlašovací údaje, pouze samotný příkaz.
2. Server ověří identitu klienta a autentizuje jej pro další operace, nebo spojení ukončí, pokud ověření neproběhlo s kladným výsledkem
3. Klient odešle žádost na vyhledání/změnu záznamu
4. Server vrátí klientovi 1 nebo více záznamů, které nalezl
5. Server vrátí návratový kód klientovi
6. Klient odešle serveru žádost o operaci *unbind*, čímž serveru oznamuje, že chce spojení ukončit
7. Server spojení uzavře

5.5 PHP jako klient pro LDAP

PHP lze využít jako LDAP klienta, potřebujete ale verzi zkompilovanou s podporou LDAP. Ke kompilaci jsou třeba klientské knihovny z projektu OpenLDAP a je nutné nastavit kompilační parametr *–with-ldap[=DIR]*. Já jsem na CentOS 4.3 využil již zkompilovaný balíček *php-common* s podporou LDAP z repozitáře *centosplus*.

Pokud chcete využít SASL autentizaci nebo SSL/TLS spojení, musíte samozřejmě použít odpovídající sestavení PHP.

PHP nabízí sadu funkcí pro práci s LDAPem, bez problémů můžete provádět dotazování i manipulaci s daty na serveru.

K připojení slouží funkce ldap_connect() a ldap_bind(). Název ldap_connect()

může být trochu zavádějící, protože tato funkce neprovádí připojení k serveru, ale jen inicializuje některé parametry připojení. Samotné připojení k serveru zajišťuje až funkce ldap_bind(). To znamená, že i když ldap_connect() vrátí resource id připojení, server na dané adrese vůbec nemusí existovat a chybu zjistíme až při použití ldap_bind().

Jednoduchý příklad komunikace se serverem v PHP:

```
// adresa serveru, pro SSL muzeme pouzit ldaps://ldapserver.cz
$ldap = ldap_connect("ldap://ldapserver.cz");
if (!$this->ldap) {
    throw new Exception("Neinicializovano pripojeni k LDAPu");
} elseif (!ldap_bind($ldap, $user, $password)) {    //pripojeni
    throw new Exception("Nepripojeno k LDAP, selhal prikaz
bind");
}


$searchResult = ldap_search($ldap, "o=myorg,c=cz","uid=" .
$user);
$info = ldap_get_entries($ldap, $searchResult);
// zjistujeme existenci hesla u prvniho vraceneho zaznamu
if (isset($info[0]['loginshell'][0])) {
    print "heslo uzivatele $user: " . $info[0]['loginshell'][0];
} else {
    throw new Exception("Nenalezen uzivatel");
}
```

Při použití SSL (ldaps) je nutné mít správně vygenerovaný certifikát na serveru a klientovi. Pokud při testování nechcete ověřovat certifikát serveru, stačí na klientovi do /etc/openldap/ldap.conf přidat řádek:

```
TLS_REQCERT never
```

Trochu nevýhoda mi připadá, že při chybě ověření certifikátu serveru PHP jen suše oznámí, že spojení se serverem nemohlo být navázáno a vůbec se nedozvíte o konkrétní chybě.

Pro použití TLS šifrování PHP vyžaduje verzi LDAP protokolu 3. To určíme v kódu přidáním definice *ldap_set_option() před funkci ldap_bind()*:

```
//inicializace
$ds = ldap_connect($LDAP_SERVER,$LDAP_PORT);
if ($ds) {
  // nastavime verzi protokolu
  if (!ldap_set_option($ds, LDAP_OPT_PROTOCOL_VERSION, 3)) {
    throw new Exception("Selhalo nastaveni verze LDAP prot.");
  }
  //spusteni TLS
  if (!ldap_start_tls($ds)) {
    throw new Exception("Ldap_start_tls selhalo");
  }
  // anonymni bind
  $bth = ldap_bind($ds);

  $searchResult=ldap_search($ldap,"o=myorg,c=cz","uid=" .
$user);
  $info = ldap_get_entries($ldap, $searchResult);
  // zjistujeme existenci hesla u prvniho vraceneho zaznamu
  if (isset($info[0]['loginshell'][0])) {
    print "heslo uzivatele $user: " . $info[0]['loginshell'][0];
  } else {
    throw new Exception("Nenalezen uzivatel");
  }
}
```

Dokumentace k použití LDAPu s PHP bohužel není nejlepší, někde úplně chybí popis funkcí. Hodně informací ale jde dohledat v diskusích u definic funkcí, které často délkou a užitečností několikanásobně převyšují obsáhlost původní chabé dokumentace. Je zde mnoho popisů, jak zprovoznit komunikaci s Microsoft Active Directory, Windows2000 serverem a podobně.

6 Šifrování přenosu dat

6.1 Odposlech dat

Cesta mezi serverem a klientem je často různorodá, používá různá přenosová média. Nikdy si nemůžeme být jisti, že komunikaci serveru a klienta nikdo nezachytává. Prakticky jediné médium, které neodposlechnete je optický kabel. Ten ale najdete v drtivé většině na páteřních linkách. Většina koncových uživatelských linek (tzv. poslední míle) je na metalické bázi. Vzhledem ke stoupající oblíbenosti bezdrátových připojení se množí případy, kdy někdo pohodlně odposlouchává komunikaci na nezabezpečené bezdrátové síti. Nejde ale jen o samotný odposlech média, útočník může odposlouchávat přímo na routeru, switchi, nebo proxy serveru, pokud se do něj dokáže nabourat, nebo mu dokonce patří.

Jako programátoři webových aplikací bychom tedy neměli spoléhat na zabezpečení trasy přenosu, protože nespravujeme linku po celé délce přenosové trasy. Proti odposlechu dat při přenosu mezi serverem a klientem se lze nejúčinněji bránit šifrováním veškeré komunikace.

6.2 Princip šifrování

Obecně se používají dva typy šifer, symetrická a asymetrická.

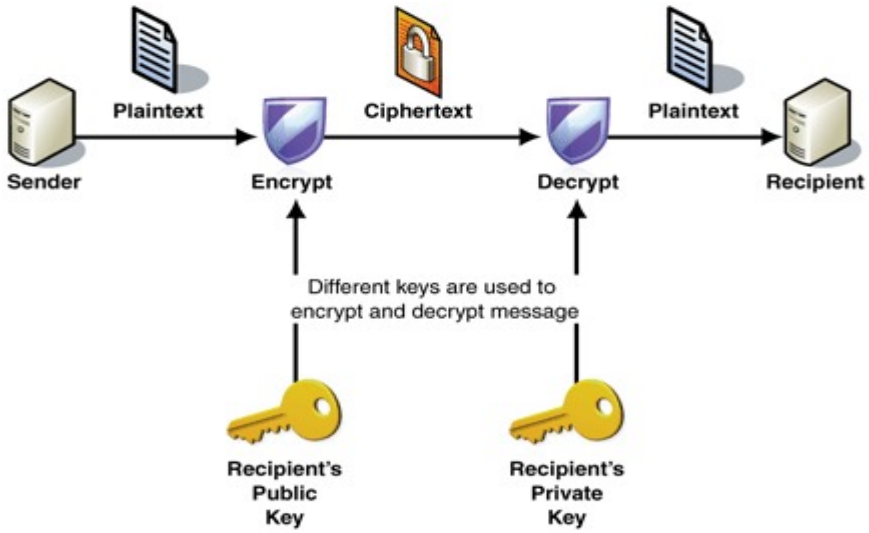
6.2.1 Symetrická šifra

Princip symetrické šifry je použití stejného klíče na straně klienta i serveru. Klient zakóduje zprávu klíčem a server ji rozkóduje tím stejným. Klíč musí mít obě strany a nesmí se ho dozvědět žádná třetí osoba. Problém je, jak takový klíč předat druhé straně, aby se ho nedozvěděl nikdo jiný. Potřebujeme nějaký zvláštní bezpečný kanál. Asi nebudete chtít klientovi posílat kurýrem disketu s klíčem pokaždé když mu budete chtít poslat šifrovaný email. Pro takové případy slouží asymetrická šifra.

6.2.2 Asymetrická šifra

Základem asymetrické šifry jsou dva páry klíčů. Klient má svůj pár a server má druhý. Používám sice označení klient a server, samozřejmě to ale může být libovolná dvojice komunikujících stran..

V každém páru klíčů je jeden veřejný a druhý soukromý. Veřejný klíč je veřejně známý, může se ho kdokoliv dozvědět. Soukromý klíč je přísně tajný a musíme ho chránit proti prozrazení. Zpráva zakódovaná veřejným klíčem se dá rozkódovat pouze pomocí soukromého klíče z daného páru. Při komunikaci si klient a server vymění veřejné klíče a odesílanou zprávu vždy šifrují pomocí veřejného klíče druhé strany. Došlé zprávy rozkódují svým soukromým klíčem.



Obr.9: Asymetrické šifrování. Zdroj: msdn.com

6.3 SSL/TLS

Pro šifrování (nejen) HTTP protokolu slouží **SSL** (Secure Socket Layer). **TLS** (Transport Layer Security) je jeho nejnovější varianta.

SSL využívá kombinace symetrické a asymetrické šifry. Asymetrická se použije pouze jako bezpečný přenosový kanál pro předání symetrického klíče a následná komunikace se šifruje jen symetrickou šifrou. Symetrická šifra je totiž výrazně méně náročná na výpočetní výkon než asymetrická.

SSL komunikace probíhá tak, že si komunikující strany vymění tzv. **handshake** zprávy. To obnáší výměnu veřejných klíčů a dohodnutí algoritmů pro šifrování (například DES, 3DES, CAST, IDEA). Klient **ověří identitu serveru** pomocí kontroly jeho certifikátu, volitelně lze ověřit i klienta. Integrita dat je kontrolována pomocí kontrolních součtů.

Pro použití SSL lze využít otevřenou knihovnu OpenSSL. Jako modul ji lze bez problémů použít pro Apache.

7 Útoky na bezpečnost aplikace

Zkusím nastínit několik typů útoků na bezpečnost webových aplikací využívajících autentizaci a navrhnout odpovídající opatření.

7.1 Odposlech komunikace

Pokud útočník odposlouchává komunikaci, jediná účinná obrana je šifrování dat. Lze zvolit buď jen šifrování autentizačních údajů, nebo všech přenášených dat. Problém nastává u vysokozátěžových aplikací (například internetový obchod), kdy je šifrování všech požadavků velmi náročné na výpočetní výkon.

7.2 Man in the middle

Útok *Man in the middle* útočník provede tak, že se dostane na komunikační kanál a dokáže číst i modifikovat přenášená data. Snaží se podvrhnout data takovým způsobem, aby obě komunikující strany o něm nevěděly, i když komunikují s ním a ne s požadovanou stranou. Pro útočníka pak není problém při SSL handshake poslat oběma stranám vlastní veřejný klíč a zjistit tak následně klíč pro další šifrování přenosu.

Jako ochrana slouží ověření digitálního certifikátu serveru, případně i klienta. To vyžaduje určitou zkušenost uživatele, aby při neúspěšném ověření nedůvěřoval serveru a komunikaci nepovolil. Další doporučení je používání nejnovější verze TLS protokolu.

7.3 Násilné prolomení hesla

U násilných metod prolomení hesla jde útočníkovi o to, aby vyzkoušel co nejvíce variant hesel v co nejkratším čase. Pro znemožnění takové činnosti můžeme implementovat několik ochran:

1. Časový zámek, který dovolí zadávat heslo maximálně jednou za určitý čas.
2. Dočasné zablokování účtu při opakovaném zadání chybného hesla. Uživateli oznámíme, že je účet blokován a ať to zkusí znovu za 10 minut.
3. Úplné zablokování účtu při opakovaném zadání chybného hesla. To je nejbezpečnější varianta, vyžaduje ale existenci uživatelské podpory při nechtěném zadání špatného hesla uživatelem. Můžeme to ošetřit resetováním

hesla a zasláním nového na uživatelův email. Tím ale prakticky přeneseme bezpečnost naší aplikace na zabezpečení uživatelova emailu

7.4 SQL injection

Útočník může do přihlašovacího pole zadat řetězec, který nesprávně zabezpečená aplikace interpretuje jako část SQL dotazu. V případě že heslo kontrolujeme v databázi pomocí SQL dotazu, hrozí tak možnost získání plného přístupu k databázi.

Jako ochrana slouží ošetření veškerých dat, která od uživatele dostaneme, tedy nejen přihlašovacích údajů, paltí to o celé webové aplikaci. Využijeme speciální „escape“ funkce, nebo předpřipravených SQL dotazů (Prepared SQL Statements). Jednoduchá funkce pro ošetření uživatelského hesla nebo jména by mohla vypadat i takto:

```
<?php
$heslo = preg_replace ('/[^a-z0-9]/', '', $_POST['heslo']);
?>
```

Pro větší zabezpečení je užitečné místo SQL dotazů využít databázové procedury a restriktivní omezení práv k databázi, které aplikaci přidělíme.

7.5 Cross site scripting

Cross site scripting (XSS) je založený na Javascriptu a cookies. Útočník využije libovolného slabého místa v aplikaci, které umožní vložit Javascriptový kód do stránky. Často k tomu dochází v návštěvních knihách, fórech a podobných uživatelům přístupných místech. Útočníkův kód pomocí Javascriptu přečte cookie libovolného uživatele a odešle ji na svůj vlastní server k zachytávání dat.

Pokud v cookie ukládáme autentizační identifikátor s dlouhodobou platností, má útočník vyhráno a může se vydávat za přihlášeného uživatele. Tato technika se označuje jako „Session Stealing“ (ukradení sezení). To samé lze udělat, pokud je identifikátor připojen v URL adrese.

Obrana:

- Ošetření veškerých vstupů, které od uživatelů dostaneme

- Minimalizace doby platnosti přihlašovacího identifikátoru, který ukládáme do cookie, nejlépe ho generovat nový při každém požadavku uživatele
- Identifikátor generovaný při každém HTTP požadavku svážeme s IP adresou uživatele. Jakmile útočník použije ukradený identifikátor, neprojde kontrolou IP adresy

7.6 Sociální inženýrství

Sociální inženýrství nemá téměř nic společného s technickým zabezpečením aplikace. Je prováděno oklamáním uživatele a vylákáním jeho přihlašovacích údajů. V poslední době se pro tuto činnost objevil moderní termín „Phishing“. Útočníci například nabídnou uživateli falešnou stránku přihlášení do aplikace, nebo se snaží vydávat za administrátory aplikace, kteří potřebují „ověřit“ jeho údaje.

Jediná obrana je vzdělávat uživatele webových aplikací, učit je jak správně odhadovat riziko a nenechat se nachytat.

7.7 Obecné zásady

Obrana před útoky celkově spočívá v zabezpečení každého článku webové aplikace. Je potřeba zabezpečit firewall, operační systém, webserver, databázový server, LDAP server a všechny další komponenty a komunikační trasy mezi nimi. Bezpečnost druhé poloviny aplikace, uživatele a jeho počítače, ovlivníte bohužel těžko. Pomůže jen osvěta a předávání zkušeností.

8 Příklady metod autentizace podle typu nasazení

Pokusím se navrhnout použití několika typů autentizace pro typické webové projekty.

Úroveň zabezpečení úzce souvisí s náklady na provoz aplikace a s omezením pohodlí uživatele. Je třeba posuzovat každý konkrétní případ zvlášť a vždy se zeptat, jak hodně si ceníme důvěryhodnosti daného projektu a informací, které jsou pomocí autentizace zabezpečené.

8.1 Osobní webová galerie

Předpokládáme webovou galerii, kam si uživatel nahraje své rodinné fotografie z dovolené a chce se o ně podlit se svými příbuznými. Nechce, aby fotografie byly veřejně přístupné, chce umožnit přístup jen určité skupině lidí.

Vlastnosti autentizace:

- Stránky s fotografiemi zabezpečíme uživatelským jménem a heslem, postačující může být HTTP Basic autentizace ověřovaná Apache serverem. Nemusíme používat zabezpečení obrazových souborů na serveru, stačí jen samotnou webovou stránku
- Můžeme implementovat funkci trvalého přihlášení. Stačí využít vestavěné PHP sessions a platnost identifikátoru session nastavit nemezenou

8.2 Internetový obchod

U internetového obchodu záleží na povaze dat, které o uživatelích ukládáme. Pokud riskujeme jen nechtěnou dobírkovou objednávku, zvolíme úroveň zabezpečení asi menší než když ukládáme čísla kreditních karet zákazníků.

- Použijeme vestavěné PHP sessions
- Zvážíme individuálně možnost trvalého přihlášení, můžeme ho povolit u fyzických osob, firmy raději necháme přihlašovat pokaždé
- Obchod bude vypadat důvěryhodněji, když použije šifrovaný SSL přenos alespoň pro přihlašování

- V případě ukládání citlivých dat budeme generovat přihlašovací identifikátor při každém požadavku, šifrovat přenos veškerých dat a implementujeme automatické zamykání účtu při zadání chybného hesla

8.3 Firemní aplikace pro nakládání s citlivými údaji

Může se jednat o webové databázové rozhraní, firemní webmail, intranetový server s firemním know-how, objednávkový systém nebo rozhraní k účetnímu systému. Potřebujeme vysokou úroveň zabezpečení, průnik útočníka by mohl způsobit ztrátu důvěryhodnosti firmy, poškodit firemní data, ztrátu výsadního postavení na trhu a podobně.

- Omezíme firewallem využití aplikace jen na intranet, pokud to nevadí použití aplikace
- Použijeme zabezpečené SSL připojení pro šifrování veškeré komunikace
- Ověřujeme klientské počítače pomocí digitálního certifikátu
- Pro přihlášení využijeme identifikátor v cookie svázaný s IP adresou, platnost omezíme na jednotlivé HTTP požadavky, čas vypršení sezení nastavíme minimální
- Veškerá přihlášení logujeme, blokujeme uživatelské účty po opakovaném zadání chybného hesla
- Pro správu uživatelů můžeme využít protokol LDAP, komunikaci s LDAP serverem šifrujeme
- Poučíme všechny zaměstnance o nakládání s hesly a certifikáty, o zabezpečení počítačů a přenosných notebooků, o nebezpečí virů a phishingu

9 Praktická ukázka typů webových autentizací

Vytvořil jsem aplikaci v PHP, která pro ověření uživatele může využít MySQL Databázi, nebo LDAP server.

Použité komponenty:

- Apache HTTP server, verze 2.0.58 pro Windows
- PHP 5.1.2 nainstalované jako modul Apache serveru
- MySQL databázový server, verze 4.1.20 pro Windows
- OpenLDAP server, verze 2.3 běžící na CentOS 4.3

PHP aplikace je balík tříd implementující úložiště pro hesla jako zásuvné moduly definované pomocí interface StorageAdapter:

```
// DB storage adapter - predame login do DB

$sa = new
StorageAdapter_Mysql('mysqli://root:1vakruk@127.0.0.1/auth');

// LDAP storage adapter - predame login do LDAP

$sa = new StorageAdapter_Ldap("ldap://ldapserver.lh",
"cn=root,o=myorg,c=cz", "password");
```

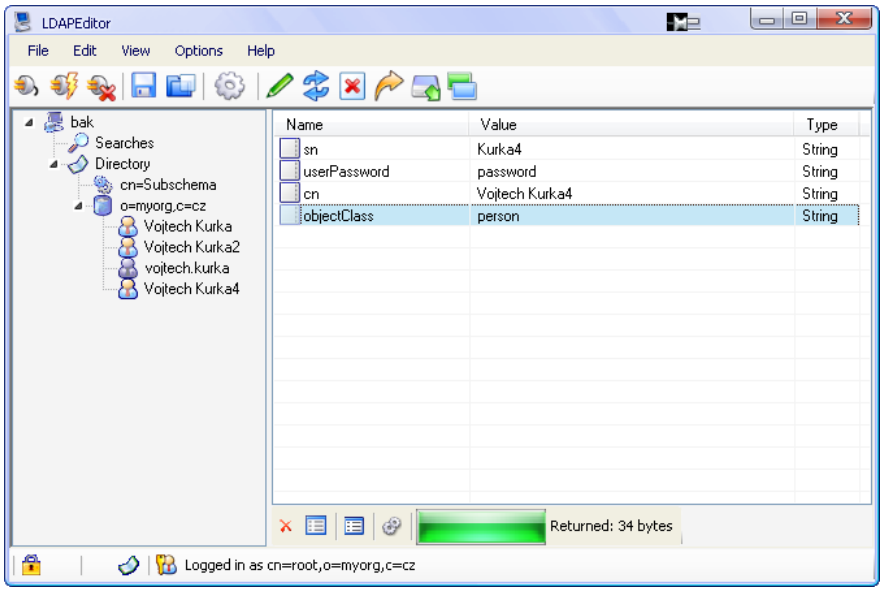
Aplikace funguje na principu vyhazování výjimek při událostech jako zadání chybného hesla, neexistujícího uživatele a podobně.

V MySQL databázi definujeme tabulku pro ukládání přihlašovacích údajů:

```
CREATE TABLE `user` (

 `userName` varchar(30) character set latin2 collate
latin2_czech_cs NOT NULL default '',

  `password` varbinary(16) NOT NULL default '',

  `permissions` varchar(200) character set latin2 collate
latin2_czech_cs default NULL,

  `lastBadLogin` datetime default NULL,

  `numBadLogin` tinyint(1) NOT NULL default '0',

  PRIMARY KEY  (`userName`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

```
INSERT INTO `user` (`userName`, `password`, `permissions`,
`lastBadLogin`, `numBadLogin`) VALUES

('vojta', 0x955db0b81ef1989b4a4dfeae8061a9a6,
'read,write,comment', NULL, 0);
```

Pro práci s LDAPem jsem použil import základních dat programem *ldapadd* (součást OpenLDAP) ze souboru ve formátu LDIF. Pro úpravy údajů a přehled nad LDAP adresářem mi dobře posloužila aplikace LDAPEditor pro Windows.



Obr.10: Program LDAPEditor

Pro data uživatele jsem použil schéma posixAccount:

```
dn: uid=vojtech.kurka,o=myorg,c=cz

objectClass: posixAccount

objectClass: top

objectClass: inetOrgPerson

homeDirectory: /

cn: Vojtech Kurka3

sn: Kurka3

uid: vojtech.kurka

gidNumber: 65534

loginShell: password

uidNumber: 4260

displayName: Vojtech Kurka3

givenName: Vojtech
```

V konkrétní implementaci autentizace by bylo vhodnější použít vlastní schéma, přesně odpovídající požadavkům zákazníka.

Implementována je HTTP Basic autentizace (s formulářem prohlížeče) a autentizace určená pro použití s PHP session s vlastním HTML formulářem. Aplikace kontroluje počet chybných zadání hesla a podle toho blokuje účet, použil jsem princip časového zámku, limitní jsou 3 chybná přihlášení.

Snažil jsem se ctít Zend Coding Standard pro PHP, komentáře jsou ve formátu PHPDoc. Struktura tříd by měla být dobře použitelná pro použití v již hotovém webovém projektu, mechanismus výjimek poskytuje variabilitu pro použití vlastních reakcí na vzniklé události.

Závěr

V práci jsem se snažil zachytit výhody a nevýhody různých typů autentizace na webu a vysvětlit její základní principy. Aplikaci pro ukázku autentizace jsem se snažil navrhnout pro reálné použití a pohodlnou integraci do webových projektů.

Největší neznámá pro mě bylo seznámení s LDAPem, jeho zprovoznění na CentOS distribuci a použití v kombinaci s SSL. Prakticky poprvé jsem zprovozňoval více aplikací na platformě Linux.

Časově nejnáročnější bylo pročtení stovek stran manuálů a jiných materiálů.

Myslím, že mě tato práce obohatila, získal jsem nový rozhled v adresářových službách, využití Apache serveru a přístupech k autentizaci. Doufám, že pomůže i jiným lidem, kteří budou hledat zdroje v češtině na podobné téma.

Seznam citací

[Apache.org, 2004, manuál k verzi 2.2] THE APACHE SOFTWARE FOUNDATION, http://httpd.apache.org/docs/2.2/mod/mod_auth_digest.html

[Netcraft.com, 2007] NETCRATFT LTD, URL*:* <*/news.netcraft.com/archives/>*

[Netcraft.com, 2006] NETCRATFT LTD, URL: <news.netcraft.com/archives/>

[Plíva V., 2006] PLÍVA, J. *Kryptologie pro praxi – zesílení hashovací funkce třídy SHA-1, URL: <crypto-world.info/klima/2006/ST_2006_07_19_19.pdf>*

Použitá literatura a informační zdroje

[1] Schlossnagle G.: *Pokročilé programování v PHP5*, Zoner Press, 2004

[2] Castagnetto, J.: *Programujeme PHP profesionálně*, Computer Press, 2002

Internet:

httpd.apache.org/docs/2.2

www.php.net

www.mysql.com

www.centos.org/docs/4

www.linuxjournal.com/node/1000105

www.root.cz

www.abclinuxu.cz

news.netcraft.com

ldap.benak.net

www.fi.muni.cz/~kas/p090/referaty/2004-jaro/skupina10/xkrejci4_ldap.html

linux456.vsb.cz/~tur038/int/diplomky/kacmarcik

quark.humbug.org.au/publications/ldap/ldap_tut.html

www.cryptofest.cz/

cryptography.hyperlink.cz/

crypto-world.info

ftp.ics.uci.edu/pub/ietf/http/rfc2617.txt

www.openldap.org/

www.netfilter.org/documentation/HOWTO//packet-filtering-HOWTO-7.html

httpd.apache.org/docs/2.2/howto/auth.html