Bezpečnost v IT

5. přednáška

Radek Janoštík

Univerzita Palackého v Olomouci

14. 3. 2024

Outline

Aktuální (kyber)bezpečnostní situace

- Sledujete zprávy z dění v kyberprostoru? Co se stalo?
- HW chyba GhostRace – chyba spekulativního vykonávání kódu
  - Možno přečíst paměť jiných procesů
- OpenSSH plánuje ukončit podporu algoritmu Digital Signature Algorithm (2025)
- Chyby ve VMWare – možnost opustit virtuální stroj
- Microsoft potvrdil krádež zdrojových kódu ruskými hackery

Protokol SSL/TLS – motivace, vývoj

- Transportní vrstva v modelu TCP/IP neřeší zabezpečení komunikace
  - Mezilehlé uzly mohou data číst (hesla, komunikace)
  - Inteligentní útočník může pozměnit data
  - Komunikující strany mohou popřít části komunikace
- Potřeba zabezpečit aplikační data
- Protokoly *Secure Socket Layer(SSL)* a *Transport Layer Security(TLS)*
  - Dva nekompatibilní protokoly
  - TLS vychází z SSL v3.0
  - TLS v1.0 je *de facto* SSL v3.1
- SSL vyvinula firma *Netscape* (Microsoft měl obdobný protokol – PCT)
  - Verze 1.0 kolem roku 1994 – nepublikována
  - Verze 2.0 (1995) – mnoho zranitelností, problémy při použití, „zastarána“ RFC-6176 (2011)
  - Verze 3.0 (1996) – brzo nahrazena TLS 1.0, *Internet Engineering Task Force(IETF)* – RFC-2246

Protokol SSL/TLS – náhled

- Slouží k zabezpečení dat z aplikačního protokolu – „mezivrstva“ (obrázek)
  - Nijak nezkoumá, nemodifikuje aplikační data
  - Zabezpečí(a zkomprimuje), přenese a „rozbalí“
- Architektura klient/server
  - Obousměrná komunikace (různé šifrovací klíče)
  - Server se vždy prokazuje certifikátem
  - Autentizace klienta certifikátem volitelná (anonymní režim)
- Zajišťuje šifrování, integritu a autorizaci(kontrolní součet dat a sdíleného tajemství)
  - Aplikační data rozdělena na fragmenty
  - Šifrování symetrickou šifrou (hybridní šifrování)
  - Data nejsou elektronicky podepsána
- Struktury popsány ve vlastním jazyce (podobnost s C), nepoužit ASN.1, BER či DER

Struktura SSL (1 / 2)

- SSL se „navenek“ tváří jako jeden protokol
- Je složen ze 4 dílčích protokolů
- *Record Layer Protocol(RLP)*
  - Zajišťuje manipulaci s daty z aplikační vrstvy
  - Komprimuje, šifruje, počítá kontrolní součty
  - Rozbaluje, dešifruje, kontroluje integritu
  - Neřeší tvorbu klíčů, stanovení algoritmů, ...
- *Handshake Protocol(HP)*
  - Navazuje komunikaci mezi stranami
  - Zajišťuje dohodu na šifrovacích, kompresních a hashovacích algoritmech
  - Autentizuje strany
  - Výměna data pro výpočet *hlavního tajemství*, odvození šifrovacích klíčů
  - Výpočet sdíleného tajemství pro kontrolní součet(*MAC secret*)
  - Příprava budoucí *protokolové svity*
  - Přenášen pomocí RLP jako aplikační protokol (zajímavost)

Struktura SSL (2 / 2)

- *Change Cipher Specification Protocol(CCSP)*
  - Jednoduchý, jeden účel (jasný z názvu), jediná zpráva
  - HP nastavuje budoucí svitu
  - Strana komunikace oznamuje „přecházím na novou svitu“
- *Alert Protocol(AP)*
  - Servisní protokol pro oznamování závad
  - Obdoba ICMP protokolu z vrstvy IP
- Obrázek SSL komunikace

SSL – parametry komunikace

- Všechna potřebná data v paměti 2× – aktuální(používána RLP), připravovaná(HP)
- Mezi klientem a serverem zřízena *relace*, může být tvořena více *spojeními*
- Data pro relaci:
  - *session identifier* – až 32B dlouhé číslo, jednoznačná identifikace relace
  - *peer certificate* – certifikát druhé strany
  - *cipher spec* – protokolová svita (šifrovací a hashovací algoritmus)
  - *master secret* – 48B sdílené tajemství (musí být utajeno)
  - *is resumable* – příznak, zda lze relaci obnovit
- Data pro spojení:
  - *ClientRandom a ServerRandom* – náhodná čísla obou stran
  - *server/client write MAC secret* – tajemství pro výpočet kontrolního součtu
  - *server/client write key* – šifrovací klíče pro obě strany
  - Inicializační vektory blokové šifry
  - Pořadové číslo přijatých a odeslaný zpráv

Record Layer Protocol

- Jádro protokolů SSL/TLS, označován jako „Vrstva SSL/TLS“
- Rozděluje aplikační data na fragmenty o max. délce 2^{14}
- Zajišťuje kompresi dat(volitelné), počítá kontrolní součet, šifruje
- Kontrolní součet ze zřetězení dat a *write MAC secret*
- Na „druhé straně“ zajišťuje opačné operace
- Hlavička RLP:
  - 1B – Typ dat – Aplikační a služební(HP, CCSP, AP)
  - 2B – Verze a podverze (3.0, 3.1, . . .)
  - 2B – Délka přenášených dat (2^{14} vs. 2^{16})
- Úvodní data HP protokolu nezabezpečena, nešifrována, nekomprimována (neproběhla dohoda)

Change Cipher Specification Protocol

- Protokol definuje pouze jednu zprávu – *CCS*
- Dává najevo, že odesílatel přešel na nové nastavení šifrování
- Další zprávy jsou již šifrované novými klíči, případně algoritmy
- Strany mohou poslat (časově) nezávisle
- Může být „uprostřed“ TCP segmentu
  - Část před zprávou `CCS` zpracována starou svitou
  - Část za zprávou `CCS` zpracována novou svitou

Handshake Protocol

- Protokol pro navázání spojení a výměnu počátečních informací
- Autentizuje server (prokazuje držení PK – dešifruje náhodné číslo)
- Možná autentizace klienta (často se nepoužívá)
- Výměna náhodných čísel pro výpočet sdíleného tajemství $\Rightarrow$ odvození klíčů a *MAC secret*
- Dva „módy“ – navázání nové relace („pomalé“) a obnovení předešlé
- HP se přenáší pomocí RLP
- Struktura HP paketu:
  - 1B – Typ zprávy
  - 3B – Délka zprávy
  - Zpráva samotná

HP – Obnovení relace

- Probíhala-li komunikace *dříve* je možné obnovit předchozí relaci
- Nemusí probíhat kompletní *handshake*, ale může se použít předchozí nastavení
- Klient zahajuje zprávou `ClientHello` obsahující *SessionID* předchozí relace
- Server odpovídá `ServerHello`, *CCS* a `Finished` a může posílat data
- Klient odpoví *CCS* a `Finished` a může posílat data
- Obnovení relace je rychlejší než navazování nové

Alert Protocol

- Služební protokol pro informování o chybách během komunikace
- Typy chyb:
  - Upozornění – komunikace může dále pokračovat
  - Fatální chyba – komunikace končí
- 2B zprávy složené z:
  - 1B Závažnost (`01` – upozornění, `02` – fatální chyba
  - 1B Kód chyby
- Např.: `close_notify(0)`, `bad_record_mac(20)`, `decompression_failure(30)`, `bad_certificate(42)`, `certificate_expired(45)` ,`unknown_ca(48)`,...
- Kompletní seznam chyb RFC-5246 str. 28

Zranitelnosti SSL

- *Padding Oracle On Downgraded Legacy Encryption(POODLE)* – MITM útok (2014)
  - Mnoho serverů podporovalo SSL 3.0 pro zpětnou kompatibilitu
  - Prostředník se *dočasně* vydává za server a donutí uživatele udělat downgrade na SSL 3.0
  - Poté, využije zranitelnost SSL 3.0 (nepočítá hash se zarovnáním), zkouší možnosti, server správná data přijme
  - Zjištění jednoho bajtu na maximálně 256 pokusů
- Útoky na kompresi – The Compression Ratio Info-leak Made Easy(CRIME) – 2012 a he Browser Reconnaissance and Exfiltration via Adaptive Compression of Hypertext(BREACH) – 2013
  - Hádání částí zpráv na základě prodloužení/zkrácení zprávy kompresním algoritmem
- Útoky na zastaralé, ale podporované protokolové svity
  - Výchozí nastavení serverů nemusí po čase reflektovat bezpečností „trendy“
  - Správci by měli reagovat a již nebezpečné svity nepodporovat
  - Nástroje na kontrolu – např. `https://www.ssllabs.com/ssltest/`

Heartbleed (2014)

- Chyba nebyla v protokolu, ale v jeho implementaci – OpenSSL
- Chyba v rozšíření *heartbeat*, které slouží k udržování spojení
- Klient posílá udržovací zprávu tvaru *délka* + *data*
  - „Jestli tady jsi, pošli mi: *4B – ahoj*“
- Server odpovídá kopií zprávy
  - „4B – ahoj“
- Klient pošle špatnou délku
  - „Pošli mi: *65536B – ahoj*“
- Server odpověděl zkopírovanou zprávou + obsah paměti dané délky
  - V paměti mohly být citlivé informace (PK, klíče pro šifrování, cookies, . . .)

Protokoly používající SSL/TLS

- „SSL verze“ aplikací mají speciální port nebo možné „povýšení“ na šifrovanou verzi
  - Např. HTTP port `80/tcp`, HTTPS port `443/tcp`
  - „Povýšení“ pomocí hlavičky `426 Upgrade Required`
- HTTPS
  - URI: `https:` – DNS jméno či IP adresa musí být v Alternativním jméně předmětu v certifikátu
  - Možné více jmen(subdomén), případně *wildcard* – `*`
  - Možné dvě úrovně autentizace – SSL/TLS(certifikáty) a HTTP Basic Auth (jméno a heslo), lze kombinovat
- POPS a IMAPS
  - Rozšíření o povýšení v RFC-2595
  - POP3 – příkaz *STLS*, IMAP4, ESMTP – příkaz `STARTTLS`
  - „pevné porty“ – POP3s 993/tcp, IMAP4 995/tcp, SMTPS 465/tcp

Webové aplikace – bezpečnost – motivace

- = Potenciálně zranitelné aplikace, které jsou volně dostupné
- Lákadla:
  - obsahují zajímavá data
  - poskytují zajímavé služby
  - firmy na nich mohou být závislé
- Útoky mohou být jednoduché, mnoho způsobů
- Nastavit (nějak) server a zprovoznit webovou aplikaci je snadné
- Mnoho jich však není udržováno, nereagují na „trendy“
- Za krátkým kódem se skrývá „velká mašinérie“
  - Neznalý programátor může způsobit spoustu problémů

HTTP-only webová aplikace

- Mějme nějakou (obyčejnou) webovou aplikaci, která ani nemusí mít přihlášení
- Proč bychom měli nasadit SSL?
- Kdokoliv na cestě může udělat MITM útok
- Např. vkládání reklamního obsahu do webů
- Sledování statistik přístupů
- U aplikací „s přihlášením“ je SSL téměř povinné
  - Heslo by mohl kdokoliv odchytit a přihlásit se za uživatele
  - Problematika šifrování hesla pomocí JavaScriptu
  - Viditelnost session-id, cookies

HTTP Basic Access Auth

- Protokol HTTP má základní podporu autentizace jménem a heslem
- Komunikace pomocí HTTP hlaviček
- Podpora v prohlížečích (uživatel zadá jméno a heslo)
- Server při potřebě autentizace odešle hlavičku `WWW-Authenticate: Basic realm="Restricted Area"`
- Klient musí do hlavičky každého požadavku přidávat jméno a heslo
  - Odesílá hlavičku: `Authorization: Basic jmeno:heslo`
  - `jmeno:heslo` je zakódováno v Base64
- Aplikace si musí zpracovávat hlavičky a rozhodnout, zda zobrazí obsah či ne
- Nevýhody: Heslo putuje v otevřené podobě, nemůže obsahovat dvojtečku
- Výhoda: Podpora webserverů (znepřístupnění části statické aplikace)

HTTP Basic Auth v PHP

- Ukázka implementace HTTP Basic Auth v PHP (zdroj: dokumentace PHP)

```php
<?php
if (!isset($_SERVER['PHP_AUTH_USER'])) {
  header('WWW-Authenticate: Basic realm="My Realm"');
  header('HTTP/1.0 401 Unauthorized');
  echo 'Text to send if user hits Cancel button';
  exit;
} else {
  echo "<p>Hello {$_SERVER['PHP_AUTH_USER']}.</p>";
  echo "<p>You entered {$_SERVER['PHP_AUTH_PW']} as your password.</p>"
}
?>
```

- Existuje mírně bezpečnější varianta HTTP Digest Access Auth (RFC-2069)
  - Kombinuje heslo s *noncí* a hashuje MD5
  - Heslo neputuje v otevřeném tvaru, ale zranitelnost MD5

Nasazení SSL

- Získání „klasického“ SSL certifikátu je sto až tiscíkorunová položka (v závislosti na CA)
- Situaci změnila CA Let’s Encrypt
  - Bezplatně vydává (již) důvěryhodné certifikáty
  - Velký důraz na automatizaci nasazení (certbot, ACME.sh)
  - Apache má vestavěný modul mod_md
  - `https://letsencrypt.org/how-it-works/`
- Privátní klíč bývá šifrován ⇒ po každém restartu služby vyžadováno heslo
  - Možné odšifrování – potom čitelný a disku
  - Pozor na nastavení oprávnění (pouze pro root: `chmod 400 server.key`)

Přihlášení uživatele

- Situace: Máme webovou aplikaci s SSL a vyplňujeme přihlašovací formulář
- Brání nám (z hlediska bezpečnosti) něco v odeslání hesla provozovateli?
- Ne, jen pouze důvěra provozovateli, že „rozumně“ pracuje s hesly
- Aplikace by měla pracovat s hesly v otevřené podobě „co nejméně“
  - Zpracování při registraci
  - Ověření při přihlášení
- V databázi by nikdy neměla být hesla v otevřené podobě
  - Unikne-li databáze z aplikace $\Rightarrow$ každý zná jejich hesla
  - Správci serveru/databáze mají přístup k heslům
- Obecně bychom (opravdu) neměli používat jedno heslo k různým aplikacím

„Rozumné nakládání s hesly“

- Varianta 1: V DB uložíme `jméno` uživatele a `hash` hesla
  - Při přihlášení spočítáme hash s hesla a porovnáme
  - Určitě lepší než plaintext
  - Jakou zvolit hashovací funkci? SHA-256? Už není považována za bezpečnou
  - Byla navržena pouze pro kontrolní součty a pro *digest*
  - Lepší funkce navržené k tomuto účelu: *bcrypt*, *scrypt*, *PBKDF2*
- Varianta 1 má problém: z uniklé DB půjde ihned poznat, že dva uživatelé mají stejné heslo
- U slabších hashovacích funkcí poměrně snadné prolomit (*rainbow tables*)
- Vygenerujeme náhodný řetězec – *sůl* pro každého uživatele
- V DB uložíme `jméno`, `sůl` a `hash(heslo+sůl)`
  - Na první pohled nelze poznat stejná hesla, prolamování je těžší
  - Např.: *PBKDF2* má sůl jako vstup

Slabá uživatelská hesla

- I skvěle zabezpečená webová aplikace může být zranitelná skrze uživatele
- Vynucování pravidel pro tvar hesla (délka, čísla, velká písmena, speciální znaky)
- Může být dvojsečná zbraň
  - Uživatelé mají tendenci mít sice silné heslo, ale na více službách
  - Dát pravidla předem na registrační formulář
- Dobrá může být kontrola výskytu ve známých slovnících
  - `https://github.com/danielmiessler/SecLists/tree/master/Passwords`
  - `https://wiki.skullsecurity.org/index.php/Passwords`
- Idea: Co tahle zpětně kontrolovat hesla v DB, zda nejsou ve slovnících?
  - Je to dobrý nápad? Proč ano? Proč ne?
- Omezování počtu přihlášení uživatele v čase (banování IP?)

Login session – session hijacking

- Po přihlášení server vygeneruje *Session Id*, které má nějakou „trvanlivost“
- Uživatelův prohlížeč posílá *Session Id* s každým požadavkem
- Server tak uživatele pozná a nemusí probíhat ověřování jménem a heslem
- Problém: Při nešifrovaném spojení můžeme odposlechnout
  - Stačí pak poslat hlavičku se *Session Id* a budeme se vydávat za uživatele
- Při krátkém *Session Id* možný bruteforce (moc se nekontroluje frekvence)
- Některé aplikace posílají *Session Id* jako parametr v URL
  - Odposlech „pohledem přes rameno“ (vyfocení)

Cross Site Request Forgery

- Uložená *Session Id* si prohlížeče pamatují do zavření/vypršení
- Uživatel se přihlásil např. na `nejaky-email.cz`
  - Získal *Session Id*, prohlížeč si jej zapamatoval
  - Při další návštěvě je přihlášen a může odesílat maily bez hesla
- Uživatel ve stejném prohlížeči otevře stránku `hack-me.cz`
- Obsahuje podvodný formulář typu:

```
<h1>Ziskejte Iphone Zdarma</h1>
<form action="https://nejaky-email.cz/sendmail.php" method="post">
    <input type="hidden" name="subject" value="Predmet emailu" />
    <input type="hidden" name="to" value="nejaka@adresa.cz" />.
    <input type="hidden" name="text" value="text emailu" />
    <input type="submit" value="Kliknete pro ziskani vyhry" />
</form>
```

Cross Site Requst Forgery

- Uživatelův prohlížeč přidá zapamatovanou *Session Id* k požadavku
- Může dojít k provedení nechtěné operaci na úplně jiném serveru
- Možné odeslat i pomocí Javascriptu, případně rovnou požadavek bez formuláře
  - Přímé JS požadavky na jinou stránku prohlížeče (již) blokují
- Obrana: *Antiforgery Token*
  - Každý formulář v aplikaci obsahuje jednorázový *token*
  - Uložen ve skrytém `input`
  - Omezená platnost (časová, použití)
  - Dnešní webové frameworky mají zabudovanou podporu (např. ASP.NET)
- Obrana: Cookies s příznakem `SameSite: Strict`
  - Takto nastavený cookie by se neměl odeslat z jiné adresy
  - Nemusí podporovat všechny prohlížeče
  - Více na: Návrh RFC, `https://web.dev/samesite-cookies-explained/`

Zabezpečení souborů

- Webový server Apache ve výchozím nastavení umožňuje zobrazit obsah adresáře
- Např.: `https://apollo.inf.upol.cz/` neobsahuje spustitelný skript
  - Po zadání adresy vytvoří „webovou stránku“ se seznamem souborů v adresáři
- Toto nastavení často zůstává zapnuto
- Možné zkoušet některé zajímavé adresáře: `/uploads`, `/images`, `/photos`, ...
- Získávání souborů pomocí cesty v URL
  - Mějme webovou aplikaci, která poskytuje soubory na adrese `app.cz/getFile.php?=./photos/photo1.php`
  - Aplikace často kontrolují pouze přítomnost soubory, nikoliv oprávnění
  - Pomocí úpravy URL můžeme získat např. hesla do DB, či zdrojové kódu
  - `app.cz/getFile.php?=../../config/config.php`
  - `app.cz/getFile.php?=../../../../etc/passwd`

Chybová hlášení

- Výpisy chybových hlášení do stránky je velmi dobré pro vývoj
  - Můžeme vidět kódy chyb
  - Stack Trace
- Při produkčním nasazení bychom neměli zobrazovat technické informace vůbec
- Pouze obecná chybová hlášení „pro lidi“
- Technická hlášení a stack trace logovat na souborový systém
- Čím méně toho útočník ví, tím hůře se mu bude útočit
  - Nebude znát strukturu aplikace (kde jsou které skripty)
  - Nemusí znát ani programovací jazyk (i přípona `.php`)
  - Neodhalovat strukturu databáze
  - Není potřeba odhalovat verzi webserveru a jazyk
- Některé frameworky mají přepínač v konfiguraci
  - Kompilace v release režimu (ASP.NET)

SQL injection

- Populární útok založený na špatném ošetřením uživatelských vstupů
- Tyto vstupy poté přímo vkládány do SQL dotazů

```
$query = "SELECT id, roles FROM users WHERE user='".$_POST["user"]."'
                                  AND hash='".hash($_POST["pass"])."'";
if (isNotEmpty(getRowFromDb($query)) { processLogin(...) }
```

- Útočník nejdříve zadá nějaký špatný znak (apostrof, středník)
- Špatně nastavený server vypíše chybovou hlášku s celým SQL dotazem
- Útočník upraví dotaz aby vrátil výsledek i bez znalosti hesla

SQL injection

- Například nastaví jméno na: `admin' --`
  - Dvě pomlčky uvozují komentář – část za nimi se bude ignorovat
- Vzniklý dotaz

```
SELECT id, roles FROM users WHERE username='admin' -- AND hash='';
```

- Špatně ošetřený dotaz získá role admina a přihlásí nás
- Poddotaz, který si zjistí hash z databáze
- Možné obejít filtrování – `' OR 1=1`
- Destruktivní dotazy – `'; DROP TABLE users;`
- Spuštění cizího kódu (Oracle – java kód, MSQQL – `xp_cmdshell`)

SQL injection – obrana

- Nepoužívat neošetřené uživatelské vstupy do SQL dotazů
- Používat SQL prepared statements (snad každý jazyk podporuje)

```
$stmt = $mysqli->prepare("INSERT INTO test(id, label) VALUES (?, ?)");
$stmt->bind_param("is", $id, $label);
```

- Hodnoty proměnných již nejsou interpretovány jako SQL
- Omezení datových typů `"is"` – integer a string
- Nezobrazovat chybová hlášky (skrytí struktury dotazů)
- ORM – většina frameworků řeší ve výchozím nastavení

Zneužití skrytých input elementů

- Některé informace si aplikace ukládají do skrytých `input` elementů a pak nekontrolují
- Př. formulář pro upravení příspěvku

```
<input type="hidden" name="post_id" value="25" />
<input type="text" name="post_text" value="Obsah prispevku" />
```

- Pouhou změnou obsahu `post_id` bychom mohli změnit příspěvek, na který nemáme oprávnění
- $\Rightarrow$ Jakýkoliv vstup (i ze skrytých elementů) musíme kontrolovat
- Extrémní případ (ne ojedinělý)

```
<input type="hidden" name="price" value="2000" />
```

- Systém bez kontroly může objednat položku za jinou cenu

FrontEnd a JavaScript

- FrontEnd aplikace by měl co nejméně prozrazovat o BackEndu
  - Nepoužívat interní objekty, ale DTO – posílat pouze potřebná data
  - Neztotožňovat názvy sloupců v DB s možností filtrace
  - Považovat všechna vstupní data z FE jako potenciálně nebezpečná
- Zdrojové kódy javascriptové části jsou veřejné
  - Každý je může studovat
  - Upravovat a zkoušet používat
  - Neměly by obsahovat citlivé informace
  - Dotazy typu: „Chtěl bych se připojit do databáze pomocí JS“
- Minifikace JS kódu jej pouze zmenšuje `https://www.minifier.org/`
- Obfuskace znesnadňuje jeho pochopení a úpravy `https://obfuscator.io/`

Cross Site Scripting (XSS)

- Zneužití nevalidování vstupu a jeho interpretace jako HTML
- Situace: Mějme návštěvní knihu s formulářem
- Útočník do formuláře vyplní:

```
Toto je ale krasna stranka!
<script>alert(document.cookie);</script>
```

- Návštěvní kniha vůbec nekontroluje vstup a výstup
  - Každému dalšímu návštěvníkovi vloží skript do DOM stránky
  - Prohlížeče daný skript interpretují a vykonají
- Cookie může být skriptem odeslán na jinou stránku
- Otravné (ztráta důvěry)

Cross Site Scripting (XSS)

- Škodlivý kód se ani nemusí zapsat do databáze
- Situace: Stránka přijímá datum jako parametr pro filtrování v JS
  `ap.cz/showByDate=23.3.2022`
  - Toto datum je někde vkládáno do DOM
- Oběti podstrčíme odkaz na
  `ap.cz/showByDate=<script>alert('HaHa');</script>`
- Kód bude vložen do stránky a proveden
- Obrana – validace uživatelského vstupu (`< > (?) # & "`)
- Překódování výstupů pomoc HTML speciální znaků
- Nastavení cookie s `HttpOnly` (nebudou dostupné pro JS)

Oblíbené redakční systémy

- Od píky se dělá velmi málo webů
- Používání frameworku, redakčních systémů (WordPress, Drupal, Joomla, . . .)
- Tyto systémy často nasazují laici
  - Často nastaveny výchozí jména a hesla, či velmi slabá
  - Nezabezpečení serverové části
  - Často zobrazují technické chybové hlášení (snadnější útok)
- Jsou to komplexní systémy $\Rightarrow$ snadno se udělá chyba
- Provozovatelé často neaktualizují („co funguje, na to nesahej“)
- V provozu obrovské množství zastaralých verzí
- Automatizované útoky (detekce CMS, verze, specializované útoky, výchozí hesla, soubory v URL, . . .)
- Obrana: Aktualizovat aplikaci a její doplňky

Doporučená četba

- Dostálek L. a kolektiv. Velký průvodce protokoly TCP/IP: Bezpečnost (2. aktualizované vydání). Computer Press, 2003. ISBN 807226849X
  - Kapitola 11 – SSL/TLS
- William Stallings. Network Security Essentials: Applications and Standards (6th Edition). Pearson, 2016. ISBN: 1-292-15485-3, 978-1-292-15485-5.
  - Kapitola 6 – Transport Layer Security
- McClure S., Scambray J., Kurtz G.: Hacking Exposed 7: Network Security Secrets and Solutions (7th. edition). CompuMcGraw Hill, 2012. ISBN 978-0071780285
  - Kapitola 10 – Web and Database hacking
- `https://owasp.org/www-community/attacks/DOM_Based_XSS`
  - XSS