Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra informačních technologií

Zabezpečení webového serveru proti odcizení dat

Bakalářská práce

Autorka: Irina Fogelzang
Studijní obor: Aplikovaná informatika

Vedoucí práce: doc. Ing. *Vladimír Soběslav*, Ph.D.
Odborný konzultant: Ondřej Kaška, eBRÁNA

Hradec Králové duben 2022

Prohlašuji, že jsem bakalářskou práci zpracovala samostatně a s použitím uvedené literatury.

V Hradci Králové dne 26.4.2022 Irina Fogelzang

Děkuji vedoucímu své bakalářské práce doc. Ing. Vladimíru Soběslavovi, Ph.D., za metodické vedení práce a čas, který mně věnoval. Zároveň vyjadřuji své poděkování také Ondřeji Kaškovi za konzultace k praktické části.

Abstrakt

Bakalářská práce se věnuje problematice komplexního zabezpečení serverů se zaměřením na úniky dat. Práce pojednává o možnostech zabezpečení, které nabízejí různé servery, s důrazem na servery Nginx a Apache. Poskytuje také přehled typických zranitelností serverů a útoků. Teoretická část bakalářské práce má za úkol především uvést čtenáře do problematiky a obsáhnout základní koncepty daného tématu. Cílem praktické části práce je pak představit možná bezpečnostní řešení a připravit podklady pro rozhodování o jejich implementaci. Tato část zohledňuje specifické požadavky podniku.

Abstract

The bachelor thesis deals with the issue of complex server security with a focus on data breaches. The thesis describes security options offered by various servers, with an emphasis on Nginx and Apache servers. It also provides an overview of typical server vulnerabilities and attacks. The theoretical part of the bachelor's thesis is aimed primarily at introducing the reader to the subject and covering the basic concepts of the topic. The aim of the practical part of the thesis is then to present possible security solutions and to prepare the basis for decision-making on their implementation. This part takes business-specific requirements into consideration.

Obsah

Seznam obrázků

1 Úvod

Každá z firem v dnešní době denně nakládá s citlivými údaji klientů a vlastním duševním vlastnictvím. Nesou odpovědnost za zabezpečení dat obsahujících tyto informace při přenosu i v klidu. Neposkytnutí odpovídajícího zabezpečení může mít za následek, že společnost ztratí pověst, klienty, finance, a dokonce může čelit obvinění. Dalším aspektem poskytování služeb na internetu je zajištění kvality a rychlosti připojení. S čím dál větší digitalizací mají uživatelé i větší očekávání. To znamená, že kybernetický útok znemožnivší legitimním uživatelům přístup ke službám může být pro firmu finančně devastující.

Podle IBM [1] byly v roce 2020 odcizené či kompromitované přihlašovací údaje nejdražšími z následků úniku dat. Osobní identifikační údaje zákazníků (Personally Identifiable Information, PII) byly nejčastěji kompromitovaným typem záznamů. Na základě výzkumu Verizonu [2] 49 % porušení v roce 2021 ohrozilo osobní údaje v technologickém sektoru, 61 % porušení se týkalo přihlašovacích údajů. U společností s méně než 1 000 zaměstnanců byly dvěma nejčastěji kompromitovanými typy údajů přihlašovací a osobní údaje. Webové aplikace jsou hlavním cílem útoků z více než 80 % porušení.

Z těchto statistik lze vyvodit, že odcizení dat prostřednictvím webových aplikací představuje velký, a přitom běžný problém. Aby se tomu předešlo, musí být jak webové aplikace firem, tak jejich webové servery hostující tyto aplikace správně nakonfigurovány a zabezpečeny. Důležité je zmínit, že rostoucí popularita práce na dálku je také rizikovým faktorem, který je potřeba při navrhování zabezpečení brát v potaz.

Cílem práce je na základě definic problémů a potřeb konkrétní firmy navrhnout nejefektivnější řešení pro zabezpečení webových serverů. Výsledkem práce by pak měla být konkrétní doporučení řešení, jež firma může dále konzultovat, případně se jimi řídit při vlastním návrhu zabezpečení.

Práce je rozdělena na dvě části. V první, teoretické části jsou popsány základní informace o webových serverech, jejich zranitelnosti a způsoby zabezpečení. Praktická část pak z těchto teoretických znalostí vychází a obsahuje odůvodnění výběru řešení i praktická doporučení pro firmu.

2 Technologie webových serverů

2.1 Definice a architektura serverů

Webový server je internetový server, který odpovídá na požadavky HTTP tím, že poskytuje obsah webových stránek a internetové služby. Termín „webový server“ může označovat jak hardware, tak software, jež se používají k hostování webových stránek.

Hardwarem rozumíme počítač, na němž jsou uloženy soubory webových stránek, které koncový uživatel chce načíst. Hardware pro webový server obvykle mívá větší RAM a HDD, rychlejší CPU.

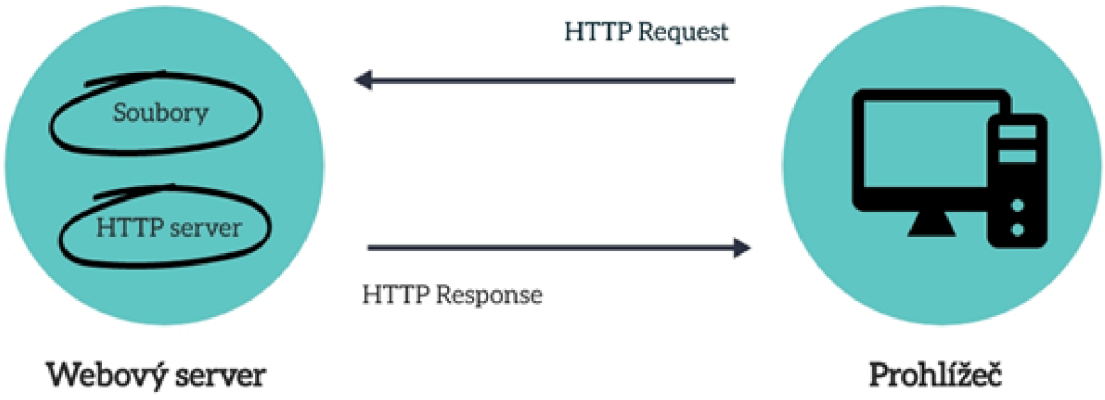
Software zahrnuje čtyři základní komponenty: operační systém, samotný http server, provádějící činnosti potřebné k hostování webových stránek, databázi a skriptovací jazyk. Nejběžněji používanou kombinaci těchto komponent představují Linux, Apache, MySQL a PHP. Tato kombinace se označuje zkratkou LAMP.

Architektura webového serveru se skládá z více parametrů [3], [4]. Jsou to:

- fyzická kapacita serveru (výpočetní výkon, úložiště a paměť),
- výkon a kvalita služeb (latence, propustnost, nízké využití paměti),
- vrstvy aplikací (typy aplikací nasazených na serveru),
- podporovaná platforma,
- operační systém,
- síťové a/nebo internetové připojení (režimy připojení a počet souběžných uživatelů, které může podporovat),
- serverová rozšíření,
- běhové prostředí pro podporu serverových rozšíření,
- aplikační služby k zajištění výkonu, bezpečnosti a provozuschopnosti aplikací.

2.2 Princip fungování

Komunikace webových služeb je založena na architektuře klient-server, kde klientem je webový prohlížeč požadující informace a serverem webový server informace poskytující. To znamená, že server dostává od prohlížeče požadavek, zpracovává ho, prohlížeč pak následně dostává odpověď. Server dohlíží na specifické porty, většinou 80 nebo 443, odkud mají požadavky přicházet. Komunikace mezi klientem a serverem probíhá pomocí protokolů – konvencí pro přenos informací mezi dvěma a více zařízeními. Standardními komunikačními protokoly jsou Hypertext Transfer Protocol (HTTP) a Hypertext Transfer Protocol Secure (HTTPS).



Obrázek 1: Schéma komunikace prostřednictvím HTTP, zdroj: vlastní zpracování

HTTP je komunikační protokol aplikační úrovní používaný pro webovou komunikaci. Funguje na principu požadavků a odpovědí, jak bylo popsáno výše. Protokol standardizuje výměnu dat mezi entitami přes počítačovou síť tím, že definuje pravidla, jež účastníci komunikace musí dodržovat. Data, která si mezi sebou účastníci komunikace posílají, jsou zapouzdřena ve formátu zprávy. HTTP zpráva sestává z povinného počátečního řádku a nepovinných hlavičky a těla zprávy [5]. Počáteční řádek definuje typ zprávy – požadavek nebo odpověď – a jeho konkrétní typ.

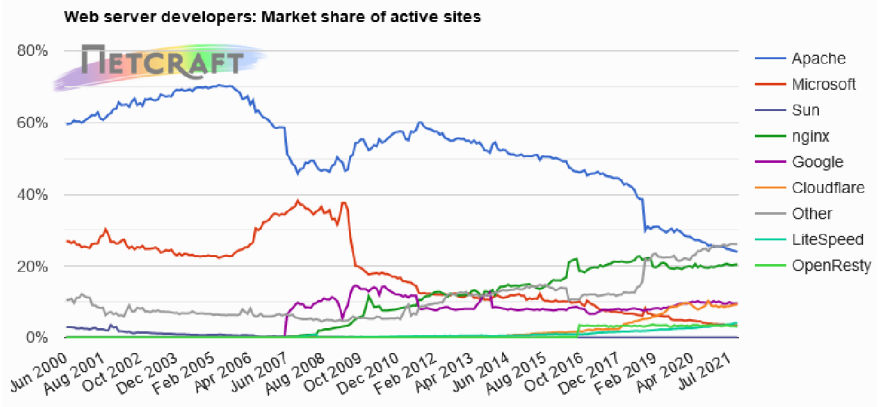
HTTP posílá informace v prostém textu, jakékoliv jiné zařízení v síti může získat takto posílaná data. Pro posílání konfidenciálních dat je nutné je šifrovat, z tohoto důvodu HTTP přestává vyhovovat. HTTP zprávy mohou být však zabezpečeny šifrovací vrstvou pomocí HTTP Secure. HTTPS je rozšíření HTTP, umožňující šifrování celé komunikace mezi odesílatelem zprávy a jejím příjemcem pomocí sekundárního kryptografického protokolu Transport Layer Security (TSL) [6].

2.3 Typy webových serverů

V současné době je k dispozici velké množství softwaru pro webové servery. Kromě rozdělení na proprietární a open source lze webové servery rozdělit do svou velkých skupin [7]:

1. Statický webový server sestává z počítače a HTTP serveru. Takový server posílá hostované soubory prohlížeči v tom stavu, v jakém je má uloženy.
2. Dynamický webový server má navíc další softwarové komponenty, často aplikační server a databázi. Aplikační server dodává business logiku a umožňuje uživatelům například interagovat s dynamickým obsahem webových stránek. Dynamický server oproti statickému aktualizuje své soubory ještě dříve, než je pošle prohlížeči.

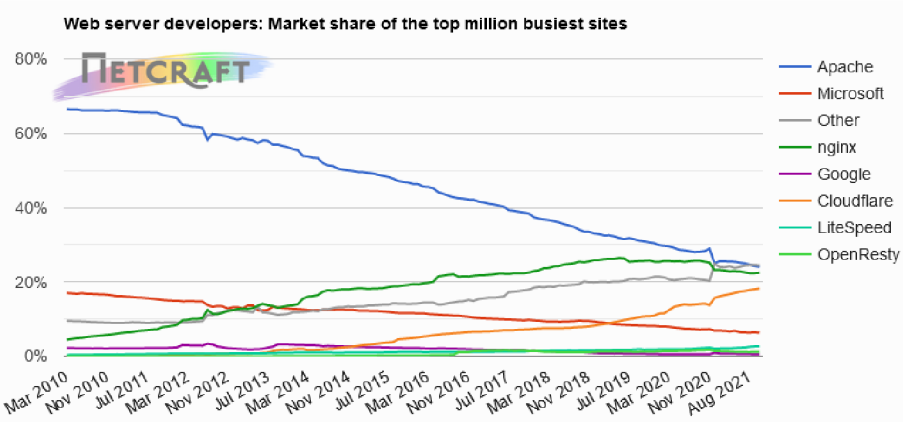
Vzhledem k početnosti softwaru a jeho vysoké míře upravitelnosti podle potřeb každého může být složité rozhodnout se pro určitý produkt. Kromě technických parametrů pomoci v rozhodování mohou i výsledky různých průzkumů. Například podle průzkumu Netcraftu [8] byl tržní podíl aktivních stránek k říjnu roku 2021 následující:



Obrázek 2: Tržní podíl aktivních stránek, zdroj: **[8]**

Ačkoliv jednu dobu to vypadalo, že Apache na trhu bude výrazně dominovat i nadále, v současnosti předbíhá Nginx pouze o 4 %. Google a Cloudflare jsou na velmi podobné hladině, kolem 10 %. Je ale potřeba zmínit poměrně prudký růst Cloudflaru v posledních letech. LiteSpeed a OpenResty, třebaže se zdají být na nízké úrovni, jsou na tom lépe než Microsoft. Zatímco tržní podíl Microsoftu již od roku 2009 stálé klesá, podíl serverů LiteSpeed a OpenResty v posledních pěti letech roste. Položka „Ostatní“ má nejvyšší procento podílu, zde je ovšem nutné brát v potaz obrovské množství serverů, jež daná položka obsahuje.

Co se týká tržního podílu prvního milionu nejrušnějších aktivních stránek, vypadá následovně:



Obrázek 3: Tržní podíl milionu nejrušnějších aktivních stránek, zdroj: **[8]**

Situace se docela podobá té na grafu č. 2 – s tím, že trendy jsou výraznější. Na nízkých úrovních jsou Google pro svou proprietární povahu a Microsoft z důvodu

závislosti na operačním systému Windows. Mezera mezi Apachem a Nginxem je tu ještě menší a ještě zřetelnější je růst popularity Cloudflaru.

Z těchto dvou grafů vyplývá, že nejrozšířenějšími webovými servery jsou:

1. Apache HTTP Server (svobodný a otevřený webový server, nejvíce se používá v průmyslu; vyhovuje nejen potřebám velkých firem, ale i jednotlivců),
2. Nginx (lehký a rychlý otevřený server s jednoduchou konfigurací a velkým množstvím pluginů, rozšiřujících jeho možnosti),
3. Cloudflare (společnost poskytuje služby CDN [Content Delivery Network, síť pro doručování obsahu], ochranu proti DDoS, bezpečný přístup ke zdrojům a DNS serverům. Služby Cloudflare fungují jako reverzní proxy pro webové stránky),
4. Google Web Server (proprietární webový server společnosti Google; využívá se tedy v rámci infrastruktury společnosti a pro poskytování vyhledávacích služeb),
5. Microsoft IIS (Internet Information Services) (proprietární software a webový server, jenž je součástí operačního systému Microsoft Windows; třetí nejpoužívanější webový server za Apachem a Nginxem),
6. LiteSpeed Web Server (proprietární webový server, kompatibilní s většinou funkcí nabízených Apachem.

V následujících kapitolách budou popsány funkčnosti nabízené dvěma nejpopulárnějšími servery a jejich řešení zabezpečení. Cílem je porovnání možností logování a monitorování, cachování a zabezpečení, včetně využití proxy.

2.4 Apache Web Server

Apache HTTP Server byl původně vyvinut Robertem McCoolem v roce 1995 a počínaje rokem 1999 je jeho vývoj řízen organizací Apache Software Foundation. Apache je známý tím, že hrál klíčovou roli v počátečním růstu internetu. Díky své dlouhodobé popularitě má silnou dokumentaci a podporuje integraci s velkým množstvím softwarů třetích stran. Apache je open source freeware vyvíjený a udržovaný otevřenou komunitou vývojářů a běží na celé řadě operačních systémů. Architektura obsahuje jádro Apache a rozšíření nazývané moduly. Ty plní různé funkce, jež uživatel může podle přání připojit pro rozšíření výchozích funkcí.

Klíčem k brzkému úspěchu Apache byl model „jeden server dělá všechno“. Se zvýšením úrovně provozu však přibývalo webových stránek a omezoval se výkon, tudíž byla konfigurace Apache pro zpracování reálného provozu obtížnější. Nicméně administrátoři Apache často volí pro jeho modularitu, sílu a všudypřítomnost.

2.4.1 Load balancing

Dnešní User Experience vyžaduje výkon a provozuschopnost. Aby se vyhovělo takovýmto požadavkům, je spuštěno více kopií stejného systému a zatížení je rozloženo mezi nimi. Další kopie systému může být převedena online s každým zvýšením zatížení. Tato technika se nazývá load balancing (česky vyvažování zátěže) a brání přetížení jednoho serveru, což může způsobit jeho zpomalení, zrušení požadavků, dokonce i zhroucení.

Nástroj pro load balancing používá jeden z algoritmů pro vyvažování zatížení. Daný nástroj přijímá požadavek od uživatele a na základě využívaného algoritmu jej směřuje na jeden z méně vytížených serverů. Tím je zajištěno, že žádný server nebude zpracovávat větší provoz, než dokáže zvládnout.

Vyrovnávání zatížení provádí tyto důležité úkoly:

- spravuje špičky provozu a zabraňuje špičkám na jednom serveru,
- minimalizuje dobu odezvy na požadavek uživatele,
- zajišťuje výkon a spolehlivost fyzických i virtuálních výpočetních prostředků,
- přidává redundanci a odolnost výpočetním prostředím.

V případě Apache vyvažování zátěže zajišťuje modul *mod_proxy_balancer*. Tento modul je součástí standardní instalace webového serveru Apache. Modul umožňuje vytvoření zátěžově vyváženého clusteru. *mod_proxy_balancer* může být různě konfigurován. Dovoluje například označit, že konkrétní server je výkonnější než ostatní (faktor zatížení), a proto zvládne převzít větší zátěž než jiné počítače v clusteru. Provoz může být vyvažován podle provozu (přenesené bajty) nebo požadavku (počet požadavků odeslaných na host).

Kromě toho existuje webový nástroj pro správu vyvažovače. Správce balancování administrátorovi umožňuje nastavit dostupnost jednotlivých serverů a změnit jejich faktor zatížení bez nutnosti restartování serveru. Tím pádem převádění serverů offline, jejich údržba a nakonfigurování nemají žádný vliv na koncového uživatele.

2.4.2 Logování a monitorování

Pro zajištění, že aplikace běží s optimálním výkonem a přesností, je potřeba mít přehled o metrikách monitorování její aktivity. Správné monitorování může odhalit mnoho užitečných informací o výkonu podkladové aplikace.

Jedním ze způsobů, jak monitorovat Apache, je využití modulu *mod_status* [9], který je ve výchozím nastavení povolen a poskytuje informace o aktivitě a výkonu serveru. Kromě sbírání údajů je také nutné mít místo pro jejich shromažďování a analýzu. K tomuto účelu lze použít platformy pro monitorování třetích stran. Tyto

platformy mohou analyzovat protokoly, jsou odesílány do syslogu. Apache ve výchozím nastavení podporuje odesílání error logu do syslogu, ale nepodporuje totéž pro access log.

2.4.3 Cachování

Cachování neboli ukládání dat do mezipaměti zrychluje poskytování obsahu ukládáním výsledků dotazů do paměti. Následné dotazy na stejné informace budou provedeny na základě dat načtených z paměti, místo aby požadavek byl znovu zpracováván. Cachování zvyšuje výkon a snižuje zátěž, což znamená, že obsluha klientů probíhá rychleji a vyžaduje menší počet zdrojů.

Apache nabízí použití cachovacího proxy serveru pomocí modulu *mod_proxy* [9] (platí pro verzi 1.3) nebo modulu *mod_cache* [9] (platí pro verzi 2.0, kde cachovací funkčnost byla od proxy oddělena). Výhodné na modulu je, že dobře pracuje s dynamickými stránkami, kterých je v dnešní době většina. Do mezipaměti ukládá pouze statický obsah, jako jsou obrázky, zatímco načítá nejnovější verze obsahu, který se v průběhu času mění. Tomu se říká kontrola cache. Nicméně dynamický obsah se obvykle nemění tak často, aby se nevyplatilo ho ukládat do mezipaměti. V tomto případě lze kontrolu cache vypnout a vynutit ukládání dynamického obsahu do mezipaměti s využitím nastavení časového limitu. Po jeho uplynutí by se měla načíst nejnovější verze obsahu ze serveru.

Dále lze využít moduly *mod_mmap_static* [9] nebo *mod_file_cache* [9] (pro Apache 1.3, respektive 2.0) pro ukládání do mezipaměti často zobrazovaných souborů, jako je například úvodní stránka webu.

2.4.4 Autentizace a autorizace

1. Základní autentizace

Základní autentizace funguje tak, že přihlašovací údaje uživatele jsou zakódovány reverzibilním algoritmem (obvykle kódování base-64) a výsledek odesílán ve formátu prostého textu (plain text) jako součást hlavičky požadavku. Protože základní autentizace je nebezpečná, měla by se používat pouze v prostředích, kde uložené dokumenty neobsahují citlivé údaje, nebo v případě, že neexistuje žádná alternativa.

Přístup pomocí základní autentizace se řídí pomocí modulu *mod_auth_basic* [9] a nástroje *htpasswd* [9].

2. Použití informací o systémovém účtu pro autentizaci na webu

Pro povolení uživatelům systému Unix autentizace přes web pomocí již přiřazených uživatelských jmen a hesel se používá modul *mod_auth* [9], kde lze nastavit */etc /passwd* jako cestu k heslu.

Přihlášení do webu pomocí systémových informací není nejbezpečnější řešení. Pokud se někomu podaří obdržet přihlašovací údaje, bude mít přístup nejen k webu, ale i k celému systému. Pokud je však web také zabezpečen přes SSL, nebo jestliže administrátor rozhodne, že toto riziko je zanedbatelné, lze tento přístup využít.

3. Vyprchávající hesla

Standardní funkce Apache neumožňují vypršení hesel. Existuje však několik řešení třetích stran, například modul Perl *Apache::Htpasswd::Perishable* [9] a handler modulu *mod_perl Apache::AuthExpire* [9].

Poslední implementuje časové limity pro „relace přihlášení“. To znamená, že uživatel se musí po určité době nečinnosti znovu autentizovat.

Použití *Apache::Htpasswd::Perishable* se vyžaduje pro kompletní zneplatnění hesla po určité době. Toto řešení přidává do souboru s hesly informace o vypršení platnosti. Do modulu *Apache::Htpasswd*, z něhož dědí, přidává další dvě metody, *expire* a *extend*, které nastavují datum vypršení platnosti hesla, respektive prodlužují dobu vypršení platnosti. Důležité je si uvědomit, že takový mechanismus je užitečný pouze v případě, že jsou hesla s prošlou platností pravidelně odstraňována ze souboru s hesly.

4. Digestní autentizace

Digestní (souhrnná) autentizace je jedna z metod, jež může webový server použít k vyjednávání přihlašovacích údajů. Rozdíl oproti obyčejné autentizaci spočívá v tom, že namísto jména a hesla je přenášena hashovaná forma těchto údajů, která je z kryptografického hlediska bezpečná.

Daný typ autentizace u Apache implementuje modul *mod_auth_digest* [9], používající hash MD5, uživatelské jméno, heslo a autentizační sféru pro kontrolu přihlašovacích údajů klienta. Tyto soubory vytváří utilita *htdigest* [9], která je součástí Apache. Protože šifrovaný hash zahrnuje sféru, hesla uložená v souborech vytvářených *htdigestem* lze použít pouze pro tuto konkrétní sféru.

2.4.5 Zabezpečení

1. Uspokojení libovolného počtu bezpečnostních metod

Apache lze pomocí směrnice *Satisfy* [9] instruovat, že uživatel potřebuje vyhovět kritériím jedné nebo všech použitých metod zabezpečení. Například za předpokladu, že se používají dvě metody zabezpečení:

- povolení přístupu pouze pro IP adresu 192.168.1.30,
- ověřování platného uživatele na základě uživatelského jména a hesla uložených v souboru *conf/htpasswd.*

Použitím *SatisfyOne* Apache bude po uživateli požadovat, aby měl IP adresu 192.168.1.30, anebo se přihlásil se správným uživatelským jménem i heslem, zatímco použije-li uživatel *SatisfyAll,* bude muset mít správnou IP adresu a zároveň se přihlásit pomocí platných uživatelských údajů.

2. Šifrování na straně klienta

Šifrováním provozu mezi serverem a klientem je přenášený obsah chráněn před třetími stranami, které mohou odposlouchávat provoz. Vrstvy zabezpečeného přenosu (Secure Sockets Layer, SSL) jsou nejběžnějším způsobem šifrování informací při přenosu.

Z legislativních a technických důvodů je instalace SSL na Apache komplikovanější než jednoduchá instalace modulů *mod_ssl* [9] nebo *Apache-SS* [9]. Konkrétní způsob instalace pak záleží na způsobu obdržení serveru Apache a na jeho verzi. Jedním z důležitých kroků je instalace knihovny OpenSSL, jejíž pomocí se generují certifikáty.

3. Předcházení Brute-Force útokům na hesla

Brute-Force útok lze poznat podle opakujících se neúspěšných pokusů o přihlášení. Pokud útočník používá nástroje na prolomení hesel, neúspěšných pokusů bude mnohem víc za určitý časový úsek, než co by bylo proveditelné manuálně. Předejít tomu lze dočasnou deaktivací možnosti přihlášení pro daného uživatele.

Prevence takových útoků není možná v rámci standardních autentizačních modulů Apache. Toho lze však dosáhnout použitím *Apache::BruteWatch* [9], který oznámí, kdy je uživatel napaden. Tento proces monitoruje logovací protokoly a upozorní administrátora na selhání ověření daného účtu, pokud autentizace selže n-krát během m minut (hodnoty nastavitelné). Administrátor pak může blokovat problematickou adresu.

4. Omezení přístupu proxy na určité URL adresy

Zamezit uživatelům v přístupu k určitým URL adresám pomocí proxy lze několika způsoby:

- blokováním podle klíčového slova,
- blokování konkrétní URL adresy,
- blokováním na základě shody vzoru regulárního výrazu.

Nezáleží na zvoleném způsobu, výsledkem bude vrácení serverem do stavu 403 Forbidden při neoprávněném pokusu o přístup na danou URL adresu.

5. Zabezpečení souborů serveru před škodlivými skripty

Aby byl webový server chráněn před škodlivými skripty, Apache navrhuje zajistit, aby žádný ze souborů nebyl zapisovatelný uživatelem *nobody* nebo skupinou *nobody* a aby citlivé soubory nebyly čitelné tímto uživatelem a skupinou.

Směrnice *User* a *Group* určují uživatele a skupinu, pod jejichž oprávněními bude webový server spuštěn. Výchozími hodnotami bývají často právě *nobody* a *nobody*. Protože vše běží s těmito oprávněními, všechny soubory nebo adresáře, které jsou přístupné tomuto uživateli a/nebo skupině, budou přístupné i pro skripty spuštěné na serveru. Doporučuje se tedy vytvořit zcela nového serverového uživatele a skupinu, a to z důvodu většího přehledu o jejich oprávněních. V ideálním případě by žádné soubory na serveru neměly být ve vlastnictví nebo zapisovatelné pro uživatele serveru.

6. Omezení seznamu metod přístupných uživateli

Často je žádoucí poskytnout obecný přístup k jedné nebo více metodám HTTP, zatímco ostatní omezovat. Jako příklad může sloužit povolení metody GET pro všechny uživatele, ale metody POST pouze pro přihlášené uživatele a/nebo uživatele s určitými právy. Je možné vyžadovat autentizaci uživatele podle metody pomocí směrnic *Limit* a *LimitExcept*.

7. Odvracení útoků DoS

Modul *mod_evasive* [9] detekuje, když jeden klient zadává mnoho požadavků v krátkém časovém období, a odmítá další požadavky ze strany tohoto klienta.

Konfigurace modulu klade na požadavky dvě omezení. Za prvé hodnota uložená do směrnice *DOSPageCount* určuje, kolikrát jedna klientská adresa může zadat požadavek na stejnou URL za jednu sekundu, aniž by byla zablokována. Za druhé hodnota uložená do směrnice *DOSSiteCount* určuje, kolik URL za jednu sekundu může požadovat jedna klientská adresa, aniž by byla zablokována. Tato druhá hodnota by měla být vyšší, protože jedna stránka obsahující velký počet obrázků bude způsobovat větší počet požadavků ze strany jednoho klienta. Direktiva *DOSBlockingPeriod* nastavuje interval, po který bude klient blokován. Interval se obnoví při příštím zjištění nevyhovujících podmínek požadavků od stejného hostitele, a může se tak prodlužovat do nekonečna.

8. *mod_security* pro Apache

ModSecurity [7] byl původně vyvinut Ivanem Ristićem jako nápomocný nástroj pro zabezpečení webových aplikací, které Ristić měl tou dobou na starosti. Základním konceptem bylo softwarové řešení, umístěné před webovými aplikacemi a analyzující tok dat. První podobou řešení byl plug-in pro server Apache. Roku 2002 byla vydána open source verze ModSecurity, jež rychle získala na popularitě.

Zprovoznění probíhá jednoduše stažením souborů z webové stránky modulu *mod_security* a jeho následnou instalací podle návodu. Aktuální verze *mod_security* podporuje pouze verzi 2 webového serveru Apache.

2.5 Nginx Web Server

Nginx byl navržen speciálně pro řešení omezení výkonu webových serverů Apache. Igor Sysoev začal pracovat na Nginxu v roce 2002. Za cíl pokládal vyřešit problém C10K [10] – požadavek, aby software zpracovával 10 tisíc současných připojení. To se povedlo roku 2004 díky událostmi řízené architektuře. Tato architektura zaručuje výkon a škálovatelnost serveru – vlastnosti, díky nimž začal Nginx získávat popularitu již od svého vydání.

Nginx vyniká v poskytování statického obsahu a je navržen tak, aby předával dynamické požadavky jinému softwaru za účelem jejich zpracování.

2.5.1 Load balancing

Nginx má řadu load balacing řešení, konkrétně pomocí tří protokolů [11]:

- HTTP

Modul *upstream* řídí vyvažování zátěže pro HTTP. Volitelné parametry poskytují větší kontrolu nad směrováním požadavků. Tyto parametry zahrnují váhu serveru v load balacing algoritmu (podobně jako faktor zatížení u Apache); který ze serverů je záložní; zda je server v pohotovostním režimu, je dostupný, anebo nedostupný; a jak zjistit, zda je server nedostupný.

- TCP

Load balancing pro TCP je definován modulem *stream*. Konfigurace a možnosti daného modulu se podobají těm z modulu *upstream*. Navíc je ale k dispozici řada možností, které mění vlastnosti reverzního serveru proxy pro TCP připojení. Tyto vlastnosti zahrnují například omezení ověřování SSL/TLS, časové limity a zachování připojení (keepalive).

- UDP

Load balancing pro UDP lze najít v modulu *stream*, stejně jako TCP, a je konfigurován dost podobným způsobem. K určení použití load balancingu pro UDP je potřeba pouze jedna věc – uvedení parametru *udp* v direktivě *listen*.

2.5.2 Logování a monitorování

Prvním krokem v nastavení monitorování je povolení shromažďování metrik v modulu *stub_status* [11]. Access log a error log obsahují mnoho užitečných informací vhodných pro sběr metrik. Formát access logu je úplně přizpůsobitelný. Je dobré zvážit využití funkcí syslog, kvůli splnění požadavků na výkon a bezpečnost. Zatímco soubory protokolu jsou zapisovány na disk, syslog umožňuje Nginxu odesílat data protokolu přes síťový protokol. Umožňuje to například nastavit vyhrazený systém Linux, kde se budou shromažďovat veškerá data protokolu z různých instancí Nginxu.

Kromě monitorovacích řešení třetích stran nabízí Nginx i řešení vlastní. Nginx Amplify [12] je řešení typu software jako služba (SaaS), navržené speciálně pro monitorování Nginx Open Source a Nginx Plus. Nginx Plus nabízí pokročilý monitorovací panel s dodatečnými metrikami a JSON feed, poskytující sledování všech požadavků procházejících aplikací.

2.5.3 Cachování

Konfigurace cachování v Nginxu vyžaduje deklaraci cesty a zóny mezipaměti, které se mají použít. Zóna mezipaměti se nastavuje pomocí směrnice *proxy_cache_path* [13]. Ta pak určuje cestu, kam se mají informace uložené v mezipaměti ukládat, a prostor sdílené paměti pro ukládání aktivních klíčů a metadat odpovědí. Klíč mezipaměti je hašovanou hodnotou, na jejímž základě probíhá cachování. Nginx ukládá výsledek do deklarované struktury souboru, přičemž klíč mezipaměti používá jako cestu k souboru.

Způsob, kterým se obsah ukládá do mezipaměti a načítá se z ní, lze ovládat pomocí směrnice *proxy_cache_key* [13]. Tato směrnice se spolu s libovolnou kombinací textu a proměnných vystavených serveru Nginx (cookie uživatele, hlavičky, identifikátory relací atd.) používá k vytvoření klíče mezipaměti. Kdežto pro cachování statického obsahu stačí použití názvu hostitele a URI, nastavení klíče mezipaměti pro dynamický obsah vyžaduje větší znalost způsobů interakcí uživatelů s webem. Špatné nastavení klíčů v případě dynamického kontentu může představovat bezpečnostní riziko. Konkrétně z toho může vyplynout, že se uživateli A bude zobrazovat cachovaný obsah patřící uživateli B.

2.5.4 Autentizace a autorizace

1. Základní autentizace

Základní autentizace se konfiguruje pomocí směrnic *auth_basic* a *auth_basic_user_file* [14]. Směrnice *auth_basic* přebírá parametr řetězce, který se zobrazí ve vyskakovacím okně, když na stránku přijde neautentizovaný uživatel. Směrnice *auth_basic_user_file* určuje cestu k uživatelskému souboru (*passwd*).

2. Autentizační poddotaz

V případě použití autentizačního systému třetí strany Nginx nabízí použití modulu *http_auth_request_module* [11]; modul se vztahuje na všechny požadavky a před doručením je odesílá autentizačnímu systému k ověření indentity. Směrnice *auth_request* přebírá parametr URI, který musí být lokálním interním umístěním. Na tuto adresu přijde originální požadavek, jenž bude odeslán do autentizačního systému uvedeného v *proxy_pass*.

3. Validace JSON Web Tokenů

JSON Web Tokeny [15] neboli JWT lze validovat pro účely autentizace a autorizace pouze pomocí Nginxu Plus. Nginx Plus obsahuje modul autentizace HTTP JWT. Konfigurace obsahuje směrnici *auth_jwt*, která určuje autentizační sféru, popřípadě proměnnou obsahující JWT. Dále obsahuje URI, kterou chceme zabezpečit, a cestu k souboru klíče ve standardním formátu JSON Web Key (JWK).

2.5.5 Zabezpečení

1. Řízení přístupu na základě IP adresy klienta

Řízení přístupu se provádí pomocí SSL modulů *http* nebo *stream*. Nastavuje se jednoduše pomocí směrnic *allow* nebo *deny* následovanými IP adresou nebo blokem IP adres.

2. Šifrování na straně klienta

K šifrování provozu se doporučuje využít jeden z modulů SSL, například *ngx_http_ssl_module* nebo *ngx_stream_ssl_module* [11]. Tyto moduly umožňují nastavit bezpečnostní certifikát a definovat klíč používaný k dešifrování požadavků a šifrování odpovědí.

Moduly SSL *http* a *stream* pro Nginx umožňují pokročilé šifrování a úplnou kontrolu nad přijatým handshakem SSL/TLS.

3. Upstream šifrování

Upstream šifrování se používá pro zabezpečení provozu mezi Nginxem a upstream serverem nacházejícím se mimo vlastní zabezpečenou síť nebo pro nastavení konkrétních vyjednávacích SSL pravidel. Konkrétní pravidla SSL, která má Nginx dodržovat, nastavují směrnice SSL modulu HTTP proxy. Konfigurované směrnice zajišťují například to, že Nginx ověří platnost certifikátu v upstream službě. Nebo určují, kterou verzi TLS Nginx bude používat.

4. Zabezpečení umístění

Zabezpečením umístění se rozumí povolení přístupu ke zdrojům pouze pro uživatele, kteří mají zabezpečený odkaz. Pomocí konfigurací modulu *secure_link* a směrnice *secure_link_secret* [11] se vytváří interní a veřejně přístupný blok umístění. Veřejně přístupný blok umístění */resources* vrátí 403 Forbidden, pokud identifikátor URI požadavku neobsahuje hashovací řetězec MD5, který lze ověřit pomocí tajného kódu poskytnutého směrnicí *secure_link_secret*.

Toto nastavení lze používat ve spojení s datem vypršení platnosti. V tomto případě i když uživatel má zabezpečený odkaz, ale snaží se ho otevřít po jeho zneplatnění, server vrátí 410 Gone.

5. Přesměrování HTTPS

Nešifrovaný provoz HTTP lze odesílat na HTTPS pomocí příkazu *return*, který vrací trvalé přesměrování 301 na server HTTPS na stejném hostiteli a identifikátoru URI požadavku. Příkaz může být použit pouze na určitých místech, kde se pracuje s citlivými údaji, jako jsou přihlašovací stránky.

6. Přísné zabezpečení přenosu HTTP

Pokud je přes HTTP odeslán formulář obsahující citlivé informace, přesměrování HTTPS od Nginxu nezachrání tyto informace před zachycením útokem typu Man-in-the-Middle. Užitečné však bude instruování prohlížečů, aby nikdy neposílaly požadavky přes HTTP nastavením hlavičky *Strict-Transport-Security*.

7. Uspokojení libovolného počtu bezpečnostních metod

Tato bezpečnostní funkčnost v Nginxu funguje velice podobně jako u Apache. Pro zadání instrukcí se používá směrnice *satisfy*. Za předpokladu použití dvou metod zabezpečení:

- povolení přístupu pouze pro IP adresu 192.168.1.30,
- ověřování platného uživatele na základě uživatelského jména a hesla uložených v souboru *conf/htpasswd*

Zadáním *any* do směrnice *satisfy* Nginx bude po uživateli vyžadovat buď to, aby měl IP adresu 192.168.1.30, nebo aby se přihlásil se správným uživatelským jménem i heslem. Kdežto zadá-li uživatel *all,* pak bude muset mít správnou IP adresu a zároveň se přihlásit pomocí platných uživatelských údajů.

8. Mitigace DDoS dynamické aplikační vrstvy Nginx Plus

Nginx Plus může vytvořit limit rychlosti podporující klastrové rozhraní (cluster-aware rate limit) a automaticky se plnící blocklist pro vytvoření dynamického řešení zmírňování DDoS útoku. Toto řešení používá směrnice *limit_req_zone* (synchronizovaný limit rychlosti) a *keyval_zone* (synchronizované úložiště typu klíč – hodnota) s parametrem *sync*, který synchronizuje zónu sdílené paměti s jinými počítači v aktivním-aktivním (veškeré uzly jsou aktivní zároveň) clusteru.

Při nastaveném limitu rychlosti na sto požadavků za sekundu bude klient, jenž překročí povolenou rychlost, identifikován na základě IP adresy bez ohledu na to, který uzel Nginx Plus požadavek přijme. Tato IP adresa bude následně přidána do „sin-bin". „Sin-bin" je úložištěm typu klíč – hodnota, synchronizovaným napříč klastrem; požadavky klientů, jejichž IP adresa se v úložišti nachází, podléhají velmi nízkému limitu šířky pásma. Po uplynutí nastaveného TTL je klient automaticky odebrán ze „sin-binu". Toto řešení je výhodné ve srovnání s odmítnutím požadavků tím, že umožňuje klientům nechtěně se ocitnuvším v úložišti posílat požadavky.

9. Modul Nginx Plus App Protect

Nginx App Protect poskytuje ochranu zabezpečení webových aplikací firewallem (Web Application Firewall, WAF), včetně:

- OWASP Top 10,
- kontrol odezvy,
- kontroly metaznaků,
- shody s protokolem HTTP,
- únikových technik,
- nepovolených typů souborů,
- kontroly tvaru JSON a XML,
- prevence úniku citlivých informací z aplikace.

10. ModSecurity 3.0 pro Nginx.

Předchozí verze ModSecurity, přestože fungovaly s Nginxem, trpěly špatným výkonem. Příčinou je, že ModSecurity byl zabalen do plné verze serveru Apache HTTP Server, který poskytoval vrstvu kompatibility. ModSecurity 3.0 představuje

kompletní přepis ModSecurity, který pracuje nativně s Nginxem bez vyžadování Apache.

Stejně jako Apache, i Nginx má vlastní konektor, jehož prostřednictvím knihovna *libmodsecurity* komunikuje s Nginxem.

3 Bezpečnost webového serveru

Zranitelnost je *„slabé místo aktiva nebo opatření, které může být využito jednou nebo více hrozbami“* [16].

Z výše uvedené definice vyplývá, že zranitelnost může obsahovat nejenom software a hardware webového serveru, ale i zásady zavedené pro ochranu příslušného serveru.

3.1 Aktiva

Aktivem [17] je jakákoli součást organizace, která je cenná – obvykle buď proto, že obsahuje citlivá data, nebo může být použita pro přístup k takovým informacím. To zahrnuje digitální informace, hardware, software, včetně licencí, virtuální stroje a cloudové služby.

3.2 Bezpečnostní hrozby

Bezpečnostní hrozba je *„potenciální příčina nežádoucí události, která může mít za následek poškození systému a jeho aktiv, např. zničení, nežádoucí zpřístupnění (kompromitaci), modifikaci dat nebo nedostupnost služeb“* [16].

3.2.1 Interní hrozby

Interní hrozby pocházejí od autorizované osoby, která má v rámci svých každodenních povinností snadný přístup k citlivým podnikovým datům. Primárními přispěvateli k vnitřním hrozbám jsou zaměstnanci, dodavatelé nebo externí pracovníci. Hlavní hrozby představují podvody, zneužití privilegia a/nebo zničení informací. Poměrně často ale interní bezpečnostní incident bývá způsoben lidským faktorem, jinak řečeno chybou.

Lépe zabezpečit organizaci proti vnitřním hrozbám lze zavedením zásad zabezpečení, nezanedbáváním správy zabezpečení a zvyšováním povědomí pracovníků o kyberbezpečnosti, a to včetně [18]:

- přesně definovaných oblastí zodpovědnosti každé osoby a odpovídajícím způsobem nastavených rolí,
- správně definovaných a implementovaných autorizačních metod,
- přísných zásad ohledně administrativních hesel a hesel uživatelů,
- omezení přístupu k interní síti a souborům (pouze přes interní síť),
- omezení vzdáleného přístupu (povolení připojení k interní síti přes VPN, SSH),
- omezení webových stránek povolených k návštěvě z lokálních sítí,

- opakovaných školení interních pracovníků o bezpečnostních pravidlech organizací, sociálním inženýrství, správném použití interních, ale i externích nástrojů.

3.2.2 Externí hrozby

Externí hrozby pocházejí zvnějšku organizace, především z prostředí, v němž daná organizace působí. Těmito hrozbami mohou být fyzické hrozby, sociálně-ekonomické hrozby specifické pro danou zemi, jako je aktuální sociální a ekonomická situace v zemi, bezpečnostní hrozby sítě, komunikační hrozby, lidské hrozby, jako jsou hrozby hackerů, softwarové a právní hrozby [18].

Útoky pocházející ze strany zkušených a sofistikovaných externích hackerů jsou nejtěžší na detekci a ochranu. Tito útočníci mohou najít zranitelná místa v síti nebo použít sociální inženýrství na manipulování zaměstnanců, aby se dostali přes obranu vnější sítě. To pak vede k propletení interních a externích hrozeb.

3.3 Zranitelnosti

Pro pochopení zranitelností webového serveru je důležité si uvědomit, že i zranitelnosti webových aplikací patří do tohoto kontextu, jelikož jde o rozhraní mezi uživatelem a serverem. Zranitelnost webového serveru může způsobit:

- chyba způsobená při konfiguraci nebo administraci serveru,
- chyba ve skriptu,
- chyba v jednotlivých službách (Apache, Nginx, MySQL a dalších),
- chyba zabezpečení vyskytující se v samotných webových aplikacích, s nimiž interaguje uživatel, v databázi nebo v operačních systémech.

Pro přehled může být užitečný seznam deseti hlavních webových zranitelností publikovaný OWASP neboli Open Web Security Projectem [19]. Tento seznam je založen na údajích poskytnutých různými bezpečnostními organizacemi. Zranitelnosti webového zabezpečení jsou seřazeny v závislosti na zneužitelnosti, detekovatelnosti a dopadu na software.

Aktuální seznam z roku 2021 uvádí následující zranitelnosti.

- Nefunkční řízení přístupu (Broken Access Control) může vést k neoprávněnému zveřejnění informací, úpravě nebo zničení všech dat nebo k provádění operací, na něž uživatel nemá práva.

- Kryptografické selhání (Cryptographic Failures) často vede k vystavení citlivých dat nebo kompromitaci systému.
- Injection (včetně Cross-site Scriptingu) může útočníkovi umožnit obejít ověřování, spouštět škodlivé skripty, nebo dokonce vést k úplnému ohrožení webového serveru nebo celého systému.
- Nezabezpečený design (Insecure Design) je nová kategorie se zaměřením na rizika související s nedostatky při návrhu. Zahrnuje to malé používání modelování hrozeb, bezpečných návrhových vzorů a principů i referenčních architektur.
- Chybná konfigurace zabezpečení (Security Misconfiguration), včetně dřívější kategorie pro externí entity XML (XXE), dokáže útočníkům poskytnout neoprávněný přístup k datům a funkcím systému.
- Zranitelné a zastaralé komponenty (Vulnerable and Outdated Components) mohou vést k narušení bezpečnosti citlivých údajů, a dokonce i ke zneužití celého systému.
- Selhání identifikace a autentizace (Identification and Authentication Failures) může vést k eskalaci oprávnění, jejíž dopad se může pohybovat od narušení bezpečnosti dat, úniku citlivých informací, krádeže identity až po administrativní přístup.
- Selhání integrity softwaru a dat (Software and Data Integrity Failures), zahrnující i nezabezpečenou deserializaci, je další nová kategorie, která se zaměřuje na vytváření předpokladů týkajících se aktualizací softwaru, důležitých dat a kanálů CI/CD bez ověřování integrity. Selhání integrity jsou zodpovědná za to, že se data stanou nespolehlivými, což vede k nedodržování předpisů a brání organizacím přijímat správná rozhodnutí.
- Chyby v protokolování a monitorování (Security Logging and Monitoring Failures) mohou přímo ovlivnit odpovědnost, viditelnost, upozornění na incidenty a přesnost forenzní analýzy.
- Server-Side Request Forgery je přidán kvůli názorům členů bezpečnostní komunity, avšak aktuální data nedokládají, že by šlo o závažné riziko.

3.4 Útoky na webový server

Bezpečnostní incident je *„porušení nebo bezprostřední hrozba porušení bezpečnostních politik, bezpečnostních zásad nebo standardních bezpečnostních pravidel provozu Informační a komunikační technologie*“ [16].

Útokem se rozumí *„pokus o zničení, vystavení hrozbě, změnu, vyřazení z činnosti, zcizení aktiva nebo získání neoprávněného přístupu k aktivu nebo uskutečnění neoprávněného použití aktiva“* [16]. Jednoduše řečeno útok je bezpečnostním incidentem se škodlivým úmyslem.

V případě odcizení dat jde o bezpečnostní incident, jehož výsledkem je ohrožení soukromí a zveřejnění citlivých údajů neoprávněnému subjektu.

Nejčastěji k odcizení dat dochází v oblastech služeb (včetně medicíny), financí a veřejného sektoru [20]. Externí útočníci pro neoprávněné proniknutí do systému nejčastěji využívají hacking a malware. Poměrně často ale k incidentu dojde kvůli interním pracovníkům, a to buď tím, že se chyby dopustí zcela náhodně, nebo útočníkům udají soukromé informace kvůli sociálnímu inženýrství. Ve všech odvětvích se nejčastěji prolomeným aktivem stává server [2].

3.4.1 Cíle útoků

Nejsilnější motivační faktor útočníků bývá finanční. Přihlašovací a osobní údaje jsou nejžádanějšími informacemi. Z toho vyplývá, že se cílem útoků nejčastěji stává session ID (ID relace) pro účely následné manipulace s ní.

Session ID (SID) je unikátním identifikátorem uživatele na webové stránce. Pokrok ve vývoji webových aplikací vedl k pracovním tokům, které spočívají v řadě po sobě jdoucích požadavků HTTP. Proto vznikla potřeba řetězení požadavků HTTP od stejného uživatele do relací [21].

Pokaždé, když uživatel navštíví konkrétní webovou stránku, server mu přiděluje nové SID. Nové SID pro daný web generuje i po uzavření a následném otevření prohlížeče. Po opuštění a opětovné návštěvě webu bez uzavření prohlížeče však může být SID zachováno. Session ID se ukládá nejen na serveru, ale i na webovém prohlížeči klienta. SID je možné ukládat pomocí cookies nebo URL.

Ukládání SID do URL může představovat bezpečnostní riziko, jelikož tento parametr se často používá, a může se ukládat na více místech (historie prohlížeče, cache, statistiky, HTTP hlavička apod.). Útočníkovi pak pro získání SID stačí dostat se k takové informaci pomocí např. sociálního inženýrství. V některých případech webové servery nastavují session ID životnost a po uplynutí určité doby nečinnosti uživatele

relaci ukončí a přiřadí nové session ID. Tím se dá do jisté míry zabránit zneužití získaných informací.

Jak již bylo zmíněno, server dokáže identifikovat uživatele i pomocí nástroje cookies, jenž je součástí protokolu HTTP. Cookies se ukládají na straně klienta ve formě krátkých textových souborů. Pro každou navštívenou webovou stránku se ukládá vlastní cookie, pokud to ovšem uživatel explicitně nezakázal. Jestliže je uživatel ověřen, informace souboru cookies jsou přenášeny mezi serverem a klientem v obou směrech s každým požadavkem. Bezpečnostní riziko spočívá v tom, že když server obdrží požadavek s cookie SID, zpracuje ho, aniž by zjišťoval, odkud požadavek pochází. To znamená, že hacker, používající sofistikovanější útok, jako je například Cross-Site Request Forgery, se k SID dokáže dostat. Stejně jako je tomu při ukládání SID do URL, útokům lze zabránit omezením životnosti cookies [22].

3.4.2 Typy útoků a možnosti zabezpečení

Existuje poměrně velké množství útoků na webový server. Stejně jako u zranitelnosti platí, že i útok na samotnou webovou aplikaci lze chápat jako útok na server. Zmíním část útoků, s nimiž se správce serveru potkává nejčastěji. Cílem zde je vysvětlit daný typ útoků, jak k němu dochází a jak se proti němu dá chránit.

3.4.2.1 Directory Traversal

Directory Traversal je útok dosazený využitím HTTP, jehož prostřednictvím mohou útočníci neoprávněně přistupovat k souborům nebo adresářům mimo webovou kořenovou složku. Úspěšné procházení adresářovou strukturou znamená, že se útočníci dostanou k webovým nebo uživatelským pověřením, konfiguračním souborům, databázím, popřípadě jiným webům fyzicky umístěným na stejném serveru.

Nejúčinnějším způsobem, jak zabránit zranitelnosti vůči Directory Traversalu, je se zcela vyhnout předávání uživatelského vstupu do rozhraní API souborového systému. Jestliže tak pro danou webovou aplikaci nelze učinit, pak by měly být použity zároveň dvě vrstvy obrany.

- Ověřování uživatelského vstupu před zpracováním – to musí probíhat buď pórováním obsahu s bílou listinou povolených hodnot, anebo ujištěním, že vstup obsahuje pouze povolený obsah.
- Po ověření uživatelského vstupu by aplikace měla povolit přístup k základnímu adresáři a pomocí rozhraní API systému souborů platformy kanonizovat cestu. Důležité je ověřit, že kanonizovaná cesta začíná očekávaným základním adresářem.

3.4.2.2 Man-in-the-Middle

Man-in-the-Middle (MITM) je útokem typu „sniffing“ umožňujícím útočníkovi získat citlivé informace prostřednictvím odposlouchávání komunikaci mezi koncovým uživatelem a webovým serverem. Pomocí MITM útočník dokáže zachycovat nebo upravovat zprávy vyměňované mezi uživatelem a serverem. Útočník se oběti představuje jako proxy Pokud se oběť připojí na takovou proxy, ve skutečnosti se tím připojí na online server útočníka. Potom veškerá komunikace mezi uživatelem a webovým serverem bude procházet přes útočníka.

Pro ochranu proti Man-in-the-Middle útokům se doporučuje dodržovat několik pravidel:

- používat silné šifrování WEP/WAP na bezdrátových přístupových bodech, aby bylo útočníkům znemožněno používání brute-force pro připojení k síti,
- používat VPN k šifrování komunikace v rámci místní sítě,
- používat silné přihlašovací údaje pro přístup k nastavení routeru,
- vynucovat komunikaci přes protokol HTTPS pro znemožnění využití odposlouchaných dat,
- používat ověřování založené na páru veřejného klíče pro ujištění, že komunikujete s očekávanými objekty.

3.4.2.3 DNS útok

Vzhledem k tomu, že při zabezpečení infrastruktury DNS bývá stále poměrně často zanedbáván, stává se bezpečnostním rizikem. Existuje několik typů DNS útoků:

- Domain hijacking – trafik uživatele je přesměrován na jiný webový server. Ve své podstatě je tento útok krádeží domény firmy. Útočník, který se stává nelegitimním vlastníkem domény, má přístup ke všem administrativním údajům a může je měnit podle svých potřeb.
- DNS Amplification Attack – útok zneužívá rekurzivního DNS dotazu využívaného pro vyžádání DNS mapování. Útočník posílá dotaz DNS se zfalšovanou IP adresou do otevřeného překladače DNS, čímž vyvolává odpovědi z DNS resolverů na tuto adresu. Takovým způsobem může útočník provádět DDoS útoky.
- DNS Tunneling – DNS požadavky jsou použity k implementaci příkazového a řídicího kanálu pro malware. Příchozí DNS provoz může přenášet příkazy malwaru, zatímco odchozí provoz může stahovat citlivá data nebo poskytovat odpovědi na požadavky provozovatele malwaru.

Základními prvky zabezpečení DNS jsou:

- Zpevnění rekurzivních serverů DNS pomocí rozšíření DNSSEC nebo vypnutí DNS rekurze.
- Upevnění přístupu ke správě DNS, jako jsou použití vícefaktorového ověřovaní a vyžádání povolení administrátora ke každé změně DNS záznamů.
- Pravidelná aktualizace DNS serveru.
- Izolace DNS serveru pro snížení pravděpodobnosti zasažení útoky webových aplikací. Umožní to zavřít všechny nevyužívané porty, zastavit nepotřebné služby operačního systému a pomocí firewallu povolit pouze základní služby (SSH a DNS).

3.4.2.4 XML External Entity

XML External Entity (XXE) se spoléhá na nesprávně nakonfigurovaný analyzátor XML. Některé webové aplikace přijímají od uživatele soubory ve formátech podobných XML (například HTML nebo PDF). Tyto formáty mohou importovat externí soubory, které jsou poté interpretovány na počítači, na němž je soubor XML analyzován. Pokud útočník zná umístění souboru, ke kterému nemá oprávnění, může pomocí tohoto útoku o takový soubor požádat a odpověď od serveru bude soubor obsahovat:

- přímé XXE – XML objekt je odeslán na server s příznakem (flag) externí entity,
- nepřímé XXE – po přijetí požadavku od klienta server generuje XML objekt se zadanými uživatelskými parametry [23].

Nejsnadnějším způsobem, jak se proti XXE útokům bránit, je deaktivace externích entit v analyzátoru XML. V případě, že webová aplikace nepotřebuje analyzovat XML a jemu podobné formáty, je vhodné se vyhnout použití XML a pracovat s jinými datovými formáty, například JSON. To pak zcela eliminuje možnost XXE útoků.

3.4.2.5 Buffer Overflow

Vyrovnávací paměť „přeteče“, když se program pokusí vložit do vyrovnávací paměti více dat, než může přijmout, nebo když se program pokusí umístit data do oblasti paměti mimo vyrovnávací paměť. Útok na vyrovnávací paměť může poškodit data, zhroutit program nebo způsobit spuštění škodlivého kódu.

Existují dva typy útoků daného typu:

- Stack-based – přetečení vyrovnávací paměti založené na využití paměti zásobníku, která existuje pouze během doby provádění funkce,
- Heap-based – přetečení vyrovnávací paměti založené na využití haldy zahrnuje zahlcení paměťového prostoru alokovaného pro program nad rámec paměti používané pro aktuální operace.

Kompilátory často vytvářejí náhodné hodnoty známé jako kanárci a umísťují je do zásobníku za každou vyrovnávací paměť. Tyto hodnoty varují před nebezpečím. Porovnáním hodnoty kanárku s původní hodnotou lze zjistit, zda došlo k přetečení vyrovnávací paměti. Dalšími běžnými ochrannými opatřeními jsou:

- randomizace adresového prostoru (Adress Space Randomization, ASLR) – nahodile se pohybuje po lokacích adresního prostoru datových oblastí; útoky s přetečením vyrovnávací paměti obvykle potřebují znát lokalitu spustitelného kódu, a randomizace adresních prostorů to prakticky znemožňuje,
- prevence spuštění dat (Data Execution Prevention, DEP) – označuje určité oblasti paměti jako nespustitelné, což brání spuštění kódu v takové oblasti,
- ochrana proti přepsání obslužné rutiny strukturovaných výjimek (Structured Exception Handler Overwrite Protection, SEHOP) – chrání před útokem na Structured Exception Handling (SEH), vestavěný systém pro správu hardwarových a softwarových výjimek. Zabraňuje útočníkovi, aby mohl využít techniku přepisování SEH. Na funkční úrovni je přepsání SEH dosaženo pomocí přetečení typu stack-based pro přepsání záznamu o registraci výjimky vlákna uloženého v zásobníku.

3.4.2.6 Injection

Útoky typu Injection umožňují útočníkovi dodat neočekávaný vstup do programu. Tento vstup, který je považován za součást legitimního příkazu nebo dotazu, mění provedení programu. Takový útok může být použit proti nástrojům příkazového řádku a databázím. Existuje několik druhů Injection útoků, z nichž nejznámější je SQL Injection.

- SQL Injection – řetězec SQL je v HTTP payloadu escapován, což vede k provádění dalších SQL dotazů nebo k úpravě parametrů očekávaného dotazu; útok je prováděn proti SQL interpreteru,
- Code Injection – útok je prováděn proti rozhraní příkazového řádku (CLI) volanému koncovým bodem API a je vybaven dalšími neočekávanými příkazy kvůli nedostatečné sanitizaci; tento útok tedy zneužívá nesprávně napsané API,
- Command Injection – při Command Injection koncový bod API generuje příkazy Bash, včetně požadavků od klienta; místo CLI nebo interpretera je cílem operační systém [14].

Protože existuje několik typů vektorů injekčního útoku, existuje také několik vektorů ochrany vůči nim.

Existuje mnoho technik zmírňování SQL Injection, kvůli jejímu rozšíření. Připravená prohlášení (prepared statements) jsou často považována za „první linii“

obrany proti „injekci“. Připravené příkazy fungují tak, že se nejprve zkompiluje dotaz se zástupnými hodnotami pro proměnné, a ty pak uživatel poskytne. To znamená, že zatímco určité hodnoty lze změnit, nelze změnit celý příkaz, a tedy jeho účel. Každá hlavní SQL databáze navíc nabízí vlastní metody pro lepší zabezpečení ve formě sanitace uživatelského vstupu. To zahrnuje automatické escapování znaků a znakových sad, jež představují možné riziko.

Ostatním typům Injection útoků pomůže zabránit použití principu nejnižších privilegií a přidání určitých příkazů do bílé listiny. Princip nejnižších privilegií říká, že každý prvek systému by měl mít přístup pouze k informacím a funkcím nezbytně nutným pro jeho správné fungování. Implementace tohoto principu zajistí, že i v případě průniku do systému bude kompromitováno pouze minimum komponent namísto celého systému. Pokud se vyžaduje, aby uživatel mohl posílat příkazy na server, je důležité, aby mu byla k dispozici pouze podmnožina příkazů, nikoliv celá sada.

3.4.2.7 Cross-site scripting

Cross-site scripting (XSS) útoky zneužívají skripty, které se spouštějí v prohlížečích uživatelů. Vyskytují se v důsledku nesprávné sanitace uživatelského vstupu vloženého do uživatelského rozhraní. XSS umožňuje útočníkovi spustit jakýkoli vlastní skript, aniž by k tomu potřeboval oprávnění. To mu pak umožňuje dále získávat informace z webové aplikace, svobodně komunikovat se serverem, získávat session ID a provádět phishingové útoky.

Hlavními třemi druhy tohoto útoku jsou:

- Stored XSS – skript se ukládá do databáze a může ovlivnit více uživatelů,
- Reflected XSS – skript se vkládá do URL adresy a ovlivňuje kód přímo v prohlížeči uživatele,
- DOM-based XSS – může být reflected a stored, ke spuštění skriptu využívá DOM sinks a sources. Obvykle source je DOM objekt schopný ukládat text a sink je DOM API schopné spustit skript uložený jako text. Útok probíhá zcela na straně klienta [23].

Existují dvě hlavní věci, které pomáhají chránit webovou aplikaci před XSS útoky: sanitace uživatelského vstupu a implementace politiky zabezpečení obsahu.

V ideálním případě by uživatelský vstup předávaný do DOM měl být interpretován jako textový řetězec, nikoliv jako DOM objekt. K dosažení cíle detekce řetězce můžeme vyhodnotit „string-like“ objekt (řetězce a čísla) pomocí `JSON.stringify()` a `JSON.parse()`. Řetězcové objekty však stále mohou být vyhodnoceny jako DOM nebo být na DOM převedeny. Vyhnout se tomu lze pomocí použití *innerText* namísto `innerHTML` při připojování řetězcových objektů k DOM; a to

právě z důvodu sanitace, kterou `innerText` provádí, aby se HTML tagy zobrazovaly jako řetězce.

Nicméně pokud je nutné HTML tagy vložené uživatelem interpretovat jako HTML tagy, výše zmíněný způsob nepůjde uplatnit. V takovém případě je nutné se ujistit, že nejsou přítomné škodlivé tagy ani pokusy o útěk z funkce sanitace. Přípustné HTML tagy by měly být zařazeny do bílé listiny. Při sanitaci je potřeba brát v potaz cokoliv, co převádí text na DOM nebo text na skript. Pokud je to možné, je vhodné vyhnout se následujícím rozhraním API [23]:

- element.innerHTML/element.outerHTML,
- Blob,
- SVG,
- document.write/document.writeln,
- DOMParser.parseFromString,
- document.implementation.

Politika zabezpečení obsahu (Content Security Policy, CSP) je nástroj pro konfiguraci zabezpečení. Poskytuje nastavení, která může vývojář využít k uvolnění, nebo zpřísnění bezpečnostních pravidel ohledně toho, jaký typ kódu může běžet ve webové aplikaci. CSP plně nechrání před XSS útoky, ale poskytuje kontroly jejich zmírnění. Konkrétně lze učinit následující:

- omezit počet povolených zdrojů skriptů,
- zakázat spouštění vloženého (inline) skriptu (výchozí při zapnutí CSP),
- zakázat metody poskytující interpretaci řetězců jako kódu (výchozí při zapnutí CSP).

3.4.2.8 File Inclusion

File Inclusion útoky využitím funkce „`include`" umožňují útočníkovi přistupovat k citlivým souborům nacházejícím se na webovém serveru nebo spouštět škodlivé soubory na serveru. Tento útok lze provést především na webových aplikacích se špatným mechanismem ověření vstupu:

- Local File Inclusion (LFI) – útočník používá procházení adresářů nebo podobný mechanismus k tomu, aby webová aplikace spustila soubor umístěný jinde na serveru,
- Remote File Inclusion (RFI) – zahrnutí a spuštění vzdáleně hostovaného souboru pomocí skriptu.

Minimalizovat riziko RFI útoků pomohou validace vstupu a sanitace. Hodnoty povolené do vstupních polí by měly být zařazeny do bílé listiny. Validace by měla

probíhat nejen na straně klienta, nýbrž i na straně serveru. Oprávnění k provádění operací pro adresáře, kam se nahrané soubory ukládají, musí být omezena, přičemž povoleny jsou pouze určité typy souborů a omezeny jsou i velikosti souborů pro nahrávání.

Jedním z osvědčených postupů pro zabránění LFI útokům je udržování seznamu povolených souborů, jež mohou být součástí stránky. Dalším je použití identifikátoru pro přístup k souborům, aby uživatelé viděli pouze ID souboru. Pokud je to možné, obsah souborů by měl být uložen do databáze, nevkládán na server. Nakonec lze server instruovat, aby automaticky posílal hlavičky stahování HTTP a nespouštěl hned soubory v určitém adresáři.

3.4.2.9 Cross-Site Request Forgery

Cross-Site Request Forgery (CSRF) je útok vedoucí k eskalací privilegií. Pokud uživatel přihlášený do aplikace otevře odkaz, vytvořený útočníkem, mohou být jeho jménem prováděny nechtěné akce. V tu chvíli server nedokáže rozpoznat legitimního uživatele od nelegitimního a sám uživatel si útoku ani nevšimne.

Dosáhnout cíle útočník může pomocí falšování parametrů dotazu cíleného na koncový bod HTTP GET. K distribuci útoku lze využít hypertextového odkazu, obrázku nebo jiných HTML tagů, které spouští HTTP GET požadavek automaticky.

Útočit lze i na jiné koncové body, jejichž pomocí probíhají CRUD (create, read, update, delete) operace. Takovými koncovými body jsou
POST, PUT a DELETE; z nichž se nejčastějším cílem stává POST. Útoky CSRF na POST jsou obvykle vytvářeny prostřednictvím formulářů, protože objekt HTML tag `<form>` může iniciovat požadavek POST bez nutnosti spuštění skriptu. Pokud útočník ve formuláři předpřipraví část parametrů a skryje je, pak nebudou uživateli viditelné. Avšak poté, co uživatel dodá zbytek informací (například jméno a heslo), proběhne požadavek POST s více parametry [23].

Nejlepší ochranná opatření proti CSRF jsou ověřování hlaviček a anti-CSRF tokeny. Tento typ útoku bývá často distribuován prostřednictvím e-mailu, sociálních sítí nebo jiných webových stránek. To znamená, že požadavky CSRF mají původ odlišný od legitimních požadavků. HTTP hlavičky `origin` a `referer` ukazují na původ požadavku a jsou důvěryhodné, protože nejsou změnitelné pomocí JavaScriptu. Nicméně bezporuchové to není, útočník může provést útok jakoby ze serveru, na který útočí, a tím mít stejný původ jako legitimní požadavky.

Z toho důvodu se hlavní metodou zabezpečení proti Cross-Site Request Forgery stává použití anti-CSRF tokenů. Unikátní pro daného uživatele a danou relaci CSRF token je posílán s každým požadavkem. Server, který daný token původně poslal, ověřuje, že token v požadavku nevypršel, nebyl pozměněn a je autentický.

Vedlejším velice nápomocným ochranným opatřením je použití GET požadavku bez stavu (stateless). Distribuce CSRF útoků nejčastěji probíhá pomocí GET požadavků; proto je znemožnění HTTP GET požadavkům měnit stav aplikace účinnou prevencí.

3.4.2.10 Denial of Service

Výsledkem útoků typu Denial of Service (DoS) je narušení použitelnosti aplikace pro legitimní uživatele. Základem je spotřeba hardwarových zdrojů serveru nebo klienta. Toho je dosaženo obvykle zahlcením zdrojů serveru, ovšem v některých případech lze téhož dosáhnout zneužitím chyb v kódu.

- Distributed Denial of Service, DDoS – útok pochází od více útočníků najednou. Nelegitimní požadavky mohou pocházet od velkého počtu zařízení v botnetu. Pokud se na zařízení dostal malware, může se stát součástí botnetu.
- Regex Denial of Service, ReDoS – využívá regulárního výrazu neboli regexu. Ten se používá k vyhledávání dat, která odpovídají určitému vzoru. Pokud je regulární výraz formulován tak, že jeho vyhodnocení trvá opravdu dlouho, může se stát, že serveru nezbudou zdroje na zpracování legitimních požadavků.

Při zmírňování DoS útoků patří k nejdůležitějším věcem logování a monitorování. Dále se však přístupy zabezpečení odvíjí od konkrétního typu DoS útoku. Je nutné podotknout, že žádný z přístupů nedokáže útokům typu Denial of Service předejít, může je pouze zmírnit. Navíc ochrana proti DoS může negativně ovlivnit legitimní uživatele.

Útoky ReDoS založené na regulárních výrazech lze zmírnit implementací analytických nástrojů, které budou skenovat regulární výrazy a upozorní na takové, které by mohly aplikaci uškodit. Navíc se vyplatí zakázat uživatelům vkládat regulární výrazy.

Útoky DoS pocházející od jednoho útočníka mohou být zmírněny tím, že architektura aplikace bude umožňovat jednotlivým uživatelům využívat prostředky aplikace/serveru pouze na omezenou dobu.

Služba správy šířky pásma představuje nejsnadnější způsob, jak chránit webovou aplikaci před útokem DDoS. Služba skenuje pakety, aby zjistila, jestli se zdá, že odpovídají škodlivém vzoru. Pokud je paket označen za škodlivý, nebude předán dál webovému serveru. Dodatečně lze implementovat techniku známou jako blackholing, kdy kromě hlavního aplikačního serveru existuje několik dalších. Podezřele vypadající (nebo opakovaný) provoz je odeslán na blackhole server, který oproti hlavnímu aplikačnímu serveru neprovádí žádné operace.

3.5 Obecná bezpečnostní doporučení

- Automaticky odmítat přístup k neveřejným zdrojům.
- V celé aplikaci využívat stejných mechanismů řízení přístupu.
- Vynucovat vlastnictví záznamu.
- Deaktivovat výpis adresářů webového serveru a zajistit, aby se metadata souborů a záložní soubory nenacházely ve webových kořenových adresářích.
- Logovat veškerá selhání s dostatečným uživatelským kontextem a uchovávat na dostatečně dlouhou dobu.
- Informovat správce o opakovaných poruchách.
- Omezit přístup k rozhraní API a kontroleru za účelem minimalizace škody způsobené nástroji pro automatizované útoky.
- Smazávat ze serveru stavové identifikátory relace po odhlášení uživatele. Omezit pokud možno dobu životnosti bezestavových tokenů, případně odvolávat přístup podle OAuth standardů.
- Neukládat citlivá data, pokud to není nezbytně nutné.
- Šifrovat všechna citlivá data v klidu.
- Používat aktuální a silné algoritmy šifrování, protokoly, kryptografické funkce a klíče.
- Ověřovat vstup na straně serveru pomocí bílé listiny.
- Využívat princip nejnižších privilegií.
- Vyvinout opakovatelný proces kalení, který bude používán pro konfiguraci všech prostředí.
- Odebrat, popřípadě neinstalovat nepotřebné funkce, frameworky, závislosti, knihovny a soubory.
- Odesílat bezpečnostní direktivy klientům, např. Security Headers.
- Průběžně inventarizovat verze komponent, včetně jejich závislostí na straně klienta i serveru.
- Stahovat komponenty pouze z oficiálních zdrojů přes zabezpečené odkazy.
- Implementovat vícefaktorovou autentizaci.
- Nepoužívat výchozí přihlašovací údaje, zejména pro administrátorské účty.

- Používat zabezpečeného vestavěného správce relací na straně serveru, který po přihlášení generuje nové náhodné ID relace s vysokou entropií.
- Používat digitální podpisy nebo podobné mechanismy k ověření, že software nebo data pocházejí z očekávaného zdroje a nebyly změněny.
- Mít CI/CD kanál se správnou segregací, konfigurací a řízením přístupu.
- Zajistit, aby nepodepsaná nebo nešifrovaná serializovaná data nebyla odesílána nedůvěryhodným klientům bez určité formy kontroly integrity nebo digitálního podpisu, který by detekoval manipulaci či přehrání dat.
- Vytvořit vlastní nebo přijmout externí plán reakce na incidenty a zotavení.

4 Řešení serveru pro digitální agenturu

Základ pro praktickou část představuje analýza výběru a zabezpečení webových serverů pro digitální agenturu. Ta se zabývá tvorbou a správou webových stránek, e-shopů a CRM systémů. Je to středně velká firma, z větší části pracující s klientelou z České republiky a ze Slovenska. Má však v plánu růst a rozšiřovat trh i na jiné země Evropské unie. Proto byla uzavřena spolupráce s cílem zjistit, které servery budou odpovídat požadavkům firmy a co všechno je potřeba řešit v rámci jejich zabezpečení.

4.1 Požadavky firmy na webový server

Při rozhodování, zda nějaké řešení zvolit, se musí řídit kromě best practices především požadavky firmy samotné. Technické požadavky jsou v souladu s business logikou a poskytují přehled o tom, na co firma klade důraz.

V daném případě se dá obecně říct, že se firma stará o dostupnost, spolehlivost a bezpečnost svých produktů, dále pak o škálovatelnost a jednodušší správu serverů. Níže jsou uvedeny požadavky potvrzující tvrzení.

4.1.1 Požadavky na hardware

- rozdělení strojů pro interní (firemní) a externí (klientské) věci; k tomu pak jsou odlišné požadavky na hardware:
    - interní (8 VM, 60 jader CPU, 150 GB RAM, 8 TB rychlých disků, 20 TB pomalých),
    - externí (25 VM, 190 jader CPU, 300 GB RAM, 12 TB rychlých disků);
- využití serverové farmy;
- zajištění datového centra úrovně Tier III nebo Tier IV;
- zákaz přístupu do kódu klientům, přístup pouze do administrátorského UI.

4.1.2 Požadavky na software

- operační systém: Linux,
- databáze: MySQL, Mongo,
- programovací jazyk: PHP,

- framework: Symfony,
- monitorování: Munin, NodePing.

Servery by měly podporovat tyto služby, s čímž by neměl být problém. Z hlediska bezpečnosti stačí mít aktuální verzi softwaru a kontrolovat jej ohledně známých zranitelností. V případě objevení zranitelnosti, pokud zatím není dostupná úprava od vývojářů služby, implementovat dočasné vlastní zabezpečení.

4.1.3 Požadavky na zabezpečení

- soulad s PCI [24] (zajišťuje, že informace o kreditních kartách shromážděné online jsou přenášeny a ukládány bezpečným způsobem [zabezpečení proti úniku dat]),
- soulad s ISO/IEC 27001:2013 [25] (specifikuje požadavky na tvorbu, implementací, údržbu a neustálé zlepšování systému řízení bezpečnosti informací v kontextu organizace; zahrnuje také požadavky na hodnocení a ošetření rizik informační bezpečnosti přizpůsobené potřebám organizace),
- soulad s GDPR (zajišťuje ochranu a zpracování údajů na území EU),
- šifrování (TLS, SSL, HTTPS),
- firewall a proxy,
- DDoS ochrana.

4.2 Fyzické zabezpečení

Při výběru lokace fyzického umístění serveru je nutné se zabývat následujícími otázkami, vymezujícími nejdůležitější body:

- Jaké mechanismy fyzické ochrany pro server a jeho síťové komponenty (směrovače, přepínače apod.) jsou nebo mohou být implementovány:
    - zámky,
    - kamerový systém,
    - čtečky karet,
    - spolupráce s bezpečnostní agenturou.
- Zda jsou zajištěny duplicitní zdroje elektřiny a síťového připojení.
- Zda má místnost zajištěnu kontrolu prostředí (například systémy chlazení, parozábrany).

- Jak dobře je lokalita zabezpečena proti přírodním katastrofám, pokud je jim vystavena, existuje-li pohotovostní místo mimo oblast potenciální katastrofy.

V případě, že se firma rozhodne nehostovat vlastní webové stránky nebo aplikace, může zvolit využití služeb certifikovaného server housingu. Jedná se o službu umístění vlastního serveru do cizího datového centra. Firma poskytující danou službu se postará o bezpečnost serveru a stabilní připojení k síti. Je to ideální varianta pro firmy, které se chtějí starat o svůj server samostatně, ale jejich vlastní vybavení a síťová připojení neodpovídají minimálním požadavkům na bezpečný provoz a provoz bez prostojů. Pro středně velkou firmu využití hostingu dává největší smysl.

4.3 Zálohy

Otázkou zálohování dat nezbytně nutných pro provoz, jež v jiném zdroji nelze dohledat, se zabývá každá firma. Výběr řešení záložních zdrojů (Backup solution) záleží na daném typu dat k zálohování a jejich objemu. Při návrhu plánu je nutné počítat i s tím, že firma poroste.

Data mohou být zálohována lokálně na místních systémech (on-site) anebo na vzdálených systémech (off-site). Obě varianty mají své výhody, ale i svá bezpečnostní rizika.

Lokální zálohování má tu výhodu, že data jsou fyzicky blízko. To znamená nižší požadavky na šířku odchozího pásma a rychlejší obnovení dat ze zálohy. Tyto výhody ovšem nepřevažují nad největším rizikem fyzické škody. Mít zálohované systémy na jednom místě se zálohovacím serverem znamená, že například v případě přírodní katastrofy se zvyšuje míra pravděpodobnosti ztráty obojího. To je důvod, proč zálohování na vzdálených systémech bývá vřele doporučováno.

Zálohovaná data se odesílají na vzdálené systémy, fyzicky se nacházející v odlišném místě od místního pracoviště. Může to být záložní server na jiném pracovišti, jehož vlastníkem jste, nebo služba zálohování hostovaná v cloudu. Takovéto zálohy poskytují jisté bezpečí proti katastrofickým událostem ohrožujícím data. Odesílání dat mimo interní síť je však delší a vystavuje data nebezpečí v síťovém prostoru. V tomto případě je nutné brát v potaz šifrování, rychlost internetového připojení a již zmíněnou šířku pásma. Zálohy často obsahují citlivá a důvěrná obchodní data, je důležité, aby se s daty zacházelo bezpečně a aby byla uložena způsobem bránícím neoprávněnému přístupu. Při odesílání dat mimo server je zvláště důležité zajistit jejich bezpečný přenos, nejlépe šifrováním přes TLS. Kromě toho by již uložená zálohovaná data měla být také šifrována, pokud se dlouhodobě nepoužívají (Data At Rest Encryption).

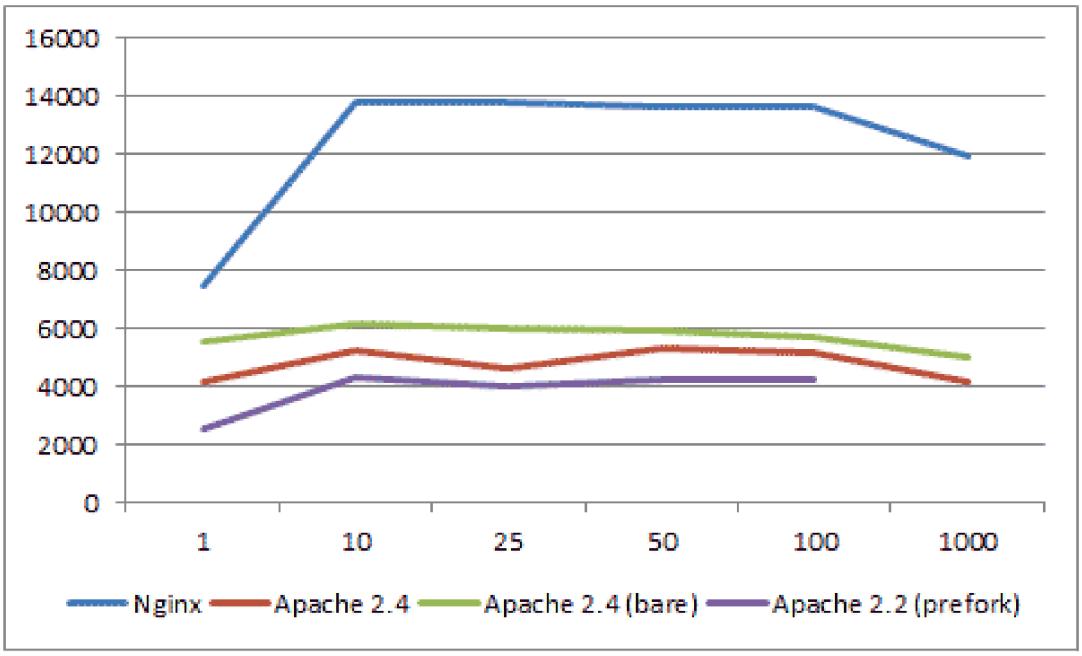
Jelikož má firma odděleny interní a externí věci, zálohovat se musí obě části zvlášť. Server se zálohami musí podporovat stávající řešení záloh. Jednou denně se vytváří kompletní snapshot clusteru a dumpy vybraných tabulek z databáze. Zálohový server se připojuje k interní síti pouze v okamžiku vytváření snapshotu. Celkem je k dispozici vždy osm až 15 záloh:

- záloha 17 týdnů stará,
- záloha 13 týdnů stará,
- záloha 10 týdnů stará,
- záloha 7 týdnů stará,
- záloha 5 týdnů stará,
- záloha 3 týdny stará,
- záloha 2 týdny stará,
- záloha 1 týden stará,
- denní zálohy za poslední týden.

4.4 Výběr webového serveru

Podle statistik webových serverů [8], [26] jsou nejpoužívanějšími servery Nginx, Apache, Cloudflare, OpenResty a LiteSpeed. Nicméně daná podkapitola se bude zabývat pouze Apachem a Nginxem. Jednoduše i z hlediska technické podpory je snadnější najít správce, kteří se vyznají v těchto dvou serverech než v méně populárních. Vzhledem ke snaze rozšiřovat byznys je pro firmu důležité používat ve své infrastruktuře prvky, které se již na trhu osvědčily.

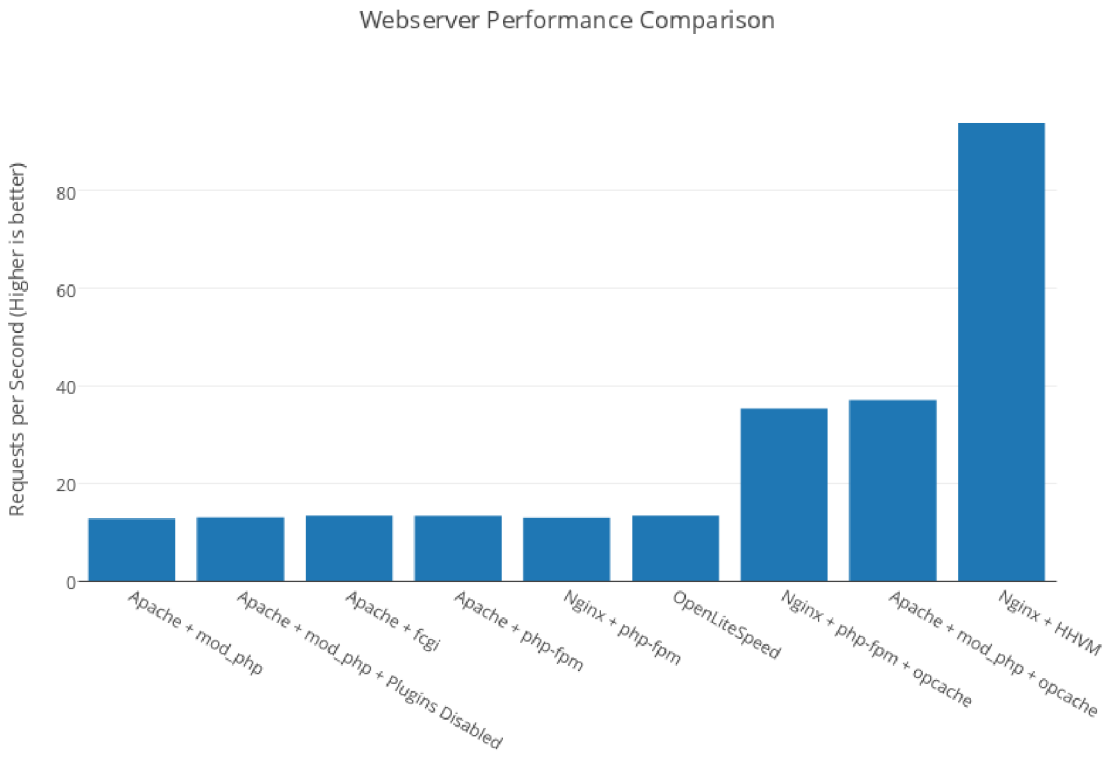
Pokud jde o statický obsah, Nginx je asi 2,5krát rychlejší než Apache na základě výsledků benchmarkového testu s až 1 000 souběžných připojení [27].



Obrázek 4: Výkon Apache 2.4 s event MPM ve srovnání s Nginxem, zdroj: **[27]**

V kapitole 3 byly zmíněny některé vlastnosti Apache a Nginxu z hlediska vyvažování zátěže, ukládání do mezipaměti a různých prvků zabezpečení. Z té kapitoly vyplynulo, že obě řešení jsou schopna dosahovat vynikajících výsledků v oblasti bezpečnosti. Stejně tak většinu základních funkcí (jako jsou právě cachování, vyrovnávání zátěže) podporují oba webové servery.

Benchmarkový test pro dynamické poskytování obsahu ukazuje, že Apache v páru s modulem PHP-FPM zvládne přibližně stejnou souběžnost jako Nginx s PHP [28]. Dále se lze podívat na tabulku ukazující dopad Apache .htaccess na výkon [29]. Nicméně je potřeba doplnit, že .htaccess umožňuje větší flexibilitu.

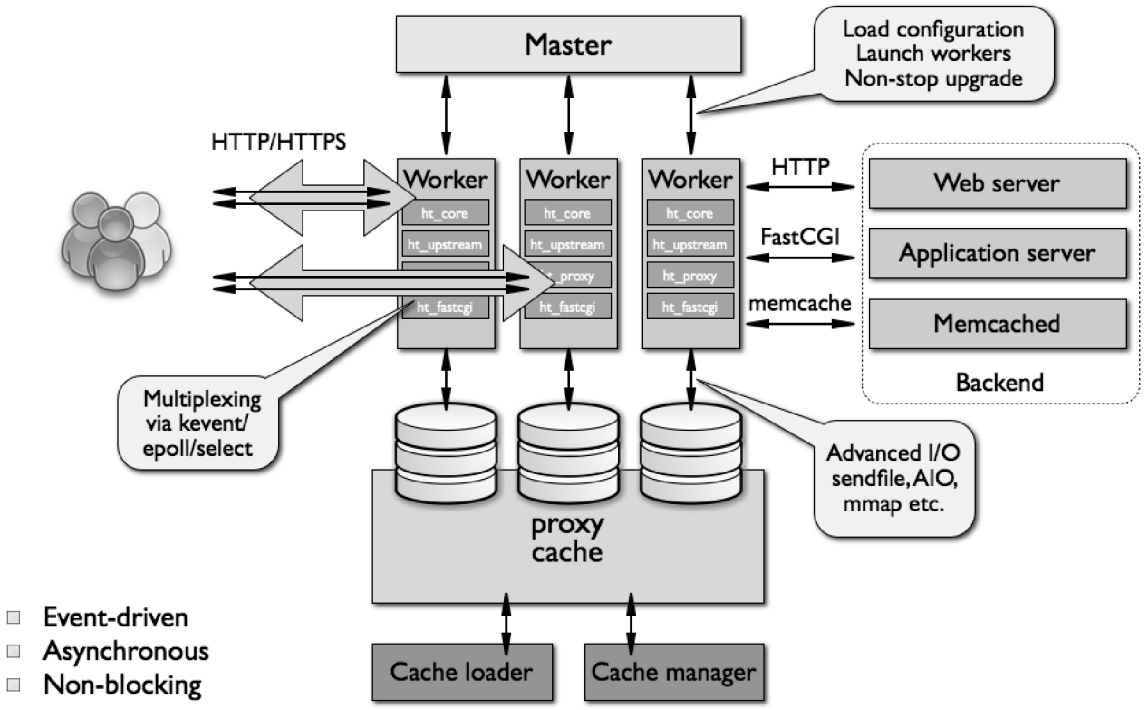


Obrázek 5: Benchmarkový test pro dynamické poskytování obsahu, zdroj: **[28]**

Requests / Hour	NGINX FS Stats	NGINX FS Reads	Apache FS Stats	Apache FS Reads	Comment
1	1	1	6	4	Single Request [Pretty much no load]
10	10	10	60	40	Ten Requests [Pretty much no load]
3,600	3,600	3,600	21,600	14,400	1 req/sec [Very low load]
144,000	144,000	144,000	864,000	576,000	40 req/sec [Moderate traffic - nothing very large]
324,000	324,000	324,000	1,944,00	1,296,000	90 req/sec [Higher traffic site - not massive]
576,000	576,000	576,000	3,456,000	2,304,000	160 req/sec [Pretty high traffic - still not massive though]

Obrázek 6: Vliv Apache .htaccess na výkon, zdroj: **[29]**

Nginx má událostmi řízenou architekturu – na rozdíl od procesově založené architektury Apache [30]. Z důvodu událostmi řízené architektury je zvládání vysokých a kolísavých zátěží předvídatelnější z hlediska využití hardwarových zdrojů.



Obrázek 7: Architektura Nginxu, zdroj: **[31]**

Pro potřeby digitální agentury by nejlepším řešením bylo využívat obojí servery: Apache jako backendový server vzhledem k tomu, že v rámci Apache může každý vývojář mít vlastní konfigurační soubor, Nginx zejména jako proxy server vzhledem k jeho schopnosti vyvažování zátěže a rychlosti zpracování statického obsahu.

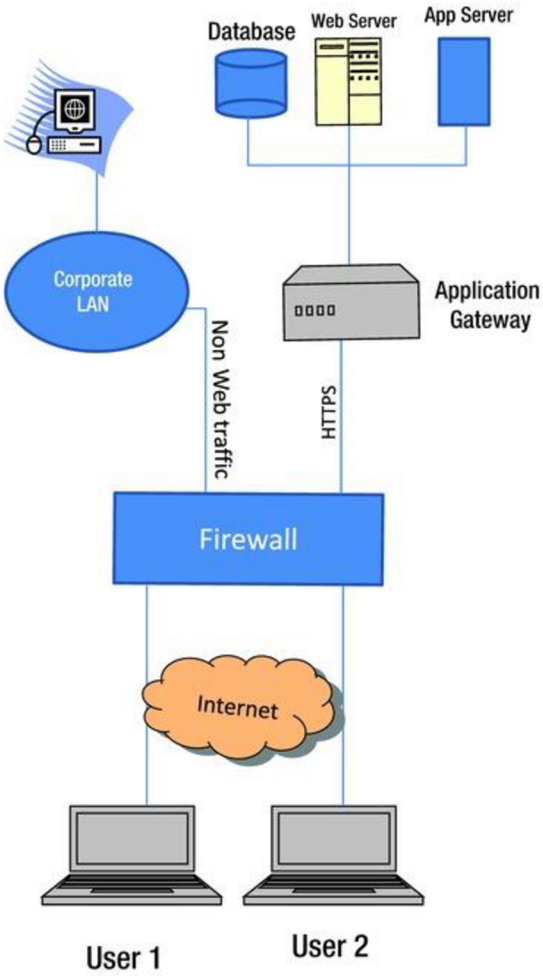
4.5 Zabezpečení vrstvy před serverem

Součástí zabezpečení serveru je přidání dodatečné vrstvy zabezpečení před samotný server. Takovou vrstvu mohou sestavovat firewall a proxy server, jejichž cílem je omezit nebo blokovat nežádoucí připojení k síti nebo ze sítě, a tím poskytnout zabezpečení před síťovými hrozbami.

Firewall v kontextu síťové bezpečnosti je zařízení, které sleduje příchozí a odchozí síťový provoz, a na základě definované sady bezpečnostních pravidel rozhoduje, zda povolí, nebo zablokuje konkrétní provoz. Může fungovat na síťové, transportní, případně aplikační vrstvě modelu OSI. Běžně bývá firewall nasazen mezi důvěryhodnou sítí organizace a nedůvěryhodnou sítí, jíž bývá zpravidla internet 19. Firewall může být realizován jako samostatné zařízení nebo softwarový program, případně jako kombinace obou.

Proxy server neboli firewall aplikační vrstvy zajišťuje nejbezpečnější typ datových spojení. Vystupuje v roli prostředníka a brání přímé komunikaci mezi místním počítačem a internetem. Místo původní IP adresy používá anonymní adresu. Proxy

server poskytuje lepší zabezpečení, protože funguje na aplikační vrstvě OSI modelu a bezpečnostní pravidla jsou založena na funkcích identifikovaných v aplikaci.



Obrázek 8: Umístění firewallu, zdroj: **[18]**

Další výhodou proxy je možnost cachování, a to jak ukládání, tak i načítání. S touto možností proxy server snižuje provoz jdoucí na server, čímž zlepšuje latenci.

Pro komplexnější řešení zabezpečení firma bude využívat jak firewall síťové vrstvy, tak i proxy server. Firewall z důvodu své rychlosti bude umístěn na místech s nejintenzivnějším provozem. Primárními účely proxy jsou v daném případě snazší řízení provozu a možnost přesunu aplikací mezi servery, dále pak jednodušší správa společných prvků (např. SSL certifikátů). Kromě toho praktické zkušenosti z proxy budou pomáhat vytvářet pravidla pro firewall.

4.5.1 Nginx jako proxy

Proxy řešení Nginx patří do vzoru tunelového agenta. To znamená, že proxy server pouze přenáší HTTPS provoz přes TCP protokol a nezná obsah provozu. Klient a server tak komunikují přímo s TLS/SSL.

Existují dva způsoby, jak používat Nginx jako HTTP proxy. Jedním je HTTP CONNECT Tunnel, který pracuje na 7. vrstvě OSI modelu, druhým pak Nginx stream, pracující na 4. vrstvě.

HTTP CONNECT není oficiálně podporován, ale může být založen pomocí modulů třetích stran, jako je modul `ngx_http_proxy_connect_module`.

Oficiální modul Nginx `ngx_stream_module` umožňuje přímo přenášet provoz přes TCP/UDP. Problém je, že proxy server nemůže získat název cílové domény z informací na úrovni 4. Pro přístup k názvu domény z HTTPS provozu bez dešifrování, používá se rozšířená adresa SNI (Server Name Indication, Indikace názvu serveru) v první klientské zprávě Hello v rámci TLS/SSL inicializace (handshake). Pro tyto účely Nginx oficiálně podporuje použití modulu `ngx_stream_ssl_preread_module`.

4.6 Nastavení a správa serveru

Co se týká bezpečnosti, nejdůležitějšími body jsou:

- zákaz SSH na veřejné IP,
- povolení administrátorských práv pro správu serveru pouze pro IP z bílé listiny,
- VPN,
- použití metody nejmenšího oprávnění (Least Privilege),
- nařízení pravidel pro hesla (minimální složitost, password manager, pravidelná změna); povinnost pro každého, kdo má administrátorská práva, pro ostatní doporučení,
- znemožnění aplikacím sahat mimo svá data.

Dále však je potřeba se zamyslet nad konkrétními kroky zabezpečení jednotlivých serverů. Možnosti zabezpečení nabízené Apachem a Nginxem byly sepsány v podkapitolách 3.5 a 3.6. Logickým pokračováním v praktické části bude vytvořit kontrolní seznam nejdůležitějších bodů a zmínit konkrétní příkazy. Všechny příkazy budou odpovídat požadavku, že systém poběží na OS Linux. Daný seznam samozřejmě nebude v žádném případě vyčerpávající. Veškeré potřebné informace však lze dohledat v oficiální dokumentaci.

4.6.1 Zabezpečení Apache

1. Zapnutí logování

Pro logování Apache nabízí modul *mod_log_config*. Formát logů lze různě modifikovat a logy mohou být zapsány přímo do souboru nebo do externího programu. Jednotlivé requesty mohou být zahrnuty do logů, nebo z nich vyloučeny na základě libovolných charakteristik díky podmíněnému logování. Verze Apache 2.0 nabízí zlepšující se daný modul *mod_logio* s možností logovat počet skutečně přenesených nebo přijatých bajtů. Navíc existují i moduly třetích stran.

2. Instalace *mod_security*

Modul *mod_security* již byl zmiňován v teoretické části práce. Tento modul obsahuje firewall, umožňuje chroot, zachycuje mnohé z nejběžnějších forem útoku a sondy (probe forms), které zasahují webové servery. Nejjednodušší způsob, jak nainstalovat tento modul, je:

```
# yum install mod_security
# /etc/init.d/httpd restart
```

3. Instalace *mod_evasive*

mod_evasive je modul třetí strany pro zabezpečení proti DoS útokům. Detekuje abnormální počet požadavků od jednoho uživatele a blokuje další požadavky z jeho strany na určitou dobu. Konfigurace obsahuje směrnice *DOSPage*, *DOSSite* a *DOSBlockingPeriod*. *DOSPage* říká, kolikrát a za jaký časový interval v sekundách musí uživatel poslat požadavek na tutéž URL adresu, aby byl zablokován. Podobně funguje *DOSSite* – s tím, že řeší požadavky na různé URL adresy. *DOSBlockingPeriod* pak určuje dobu v sekundách, po kterou bude uživatel zablokován.

4. Omezení přístupu k souborům mimo adresář web root

Pro každý disk je potřeba provést následující:

```
<Directory />
Order deny,allow
Deny from all
```

```
AllowOverride None
Options None
</Directory>
```

Poté lze pro určité sekce souborového systému tato nastavení povolit stejným způsobem.

5. Skrytí informací o serveru

Implementovat princip „zabezpečení pomocí utajení“ (security through obscurity) lze mimo jiné skrytím informací o serveru. K tomu je potřeba provést následující změny v souboru *httpd.conf*:

```
ServerSignature Off
ServerTokens Prod
```

6. Zabezpečení serverových souborů proti škodlivým skriptům

Je důležité zajistit, aby žádné skripty spuštěné na webovém serveru nemohly přistupovat k souborům, upravovat je nebo ničit. Toho lze dosáhnout tím, že se uživateli *nobody* nebo skupině *nobody* povolí zapisovat do souborů a citlivé soubory nebudou pro tyto uživatele a skupinu čitelné. Místo *nobody* lze vytvořit zcela nové uživatele a skupinu, již budou mít takto omezená práva.

7. Ochrana proti útokům na hesla hrubou silou

Není složité poznat, že útočník používá automatický nástroj na prolomení hesel. Při mnohonásobných neúspěšných pokusech o autentizaci lze předpokládat, že účet uživatele je napaden. V tom případě jeho uživatelské jméno může být dočasně blokováno.

Bohužel standardní moduly Apache tuto možnost nenabízejí. *Apache::BruteWatch* je jedním ze způsobů sledování souboru logů, který upozorní, když se nějaký účet zdá být cílem útočníka. *Apache::BruteWatch* umožňuje nastavit počet povolených neúspěšných pokusů o přihlášení za určitou dobu, e-mailovou adresu pro zasílání upozornění a časový úsek, po jehož uplynutí záznam o neúspěšném pokusu bude smazán.

8. Skrytí ETagu

ETag je hlavička obsahující některé citlivé údaje o serveru. Kromě skrývání citlivých informací před útočníky je od webových stránek elektronického obchodu standardy PCI Security Council vyžadováno, aby ETag skrývaly.

Skrýt hlavičku lze jednoduchou úpravou souboru *httpd.conf*:

```
FileETag None
```

9. Omezení změn konfiguračních nastavení

Soubory *.htaccess* umožňují uživatelům přizpůsobit chování ve svých vlastních webových adresářích. Přizpůsobení v tomto případě v podstatě znamená přepsání konfiguračních nastavení Apache. Aby se tomu zabránilo, *AllowOverride* musí být v souboru *httpd.conf* nastaveno na *None*.

10. Vypnutí SSI a CGI

Common Gateway Interface (CGI) a Server Side Includes (SSI) jsou dva standardy pro spouštění skriptů na straně webového serveru. CGI skript je jakýkoli program, který běží na webovém serveru. Soubory s povoleným SSI jsou soubory zpracovávané serverem před jejich odesláním klientovi. Soubory s povoleným SSI a CGI skripty mohou vystavit server bezpečnostním hrozbám tím, že umožňují útočníkovi zneužít jakékoli chyby, které se v nich objeví. V praxi to může znamenat přetížení serveru, získání citlivých informací, vložení škodlivých skriptů. Proto se doporučuje je vypnout nebo omezit jejich činnost prostřednictvím směrnice *Options*.

4.6.2 Zabezpečení Nginxu

1. Vypnutí nepotřebných HTTP metod

Pro webové operace jsou obvykle vyžadovány metody GET, HEAD a POST. Jakákoli jiná metoda, která nebude použita, může být bezpečně zablokována. Týká se to zejména metod TRACK a TRACE, které se někdy používají k ladění, ale lze je zneužít k získání citlivých dat o uživatelích.

Příslušné změny lze provést buď v sekci *location,* nebo *server* konfiguračního souboru Nginx.

```
location / {
limit_except GET HEAD POST { deny all; }
```

}

2. Instalace *mod_security*

Modul *mod_security* a jeho užití již byly několikrát zmíněny. Návod na instalaci na Nginx lze vyhledat na GitHub stránce modulu v záložce Wiki. Zároveň tam jsou k nalezení veškeré potřebné soubory.

3. Nastavení logů přístupů a chyb Nginxu

Logy přístupů a chyb jsou v Nginxu ve výchozím nastavení zapnuté a nacházejí se v *logs/error.log* a *logs/access.log*. Umístění a úroveň závažnosti chyb, které mají být zaznamenány, lze změnit ve směrnici *error_log* v konfiguračním souboru Nginx. Dále také lze upravit směrnici *log_format* pro změnu formátu logovaných zpráv.

4. Vypnutí *server_tokens*

Ve výchozím nastavení je direktiva *server_tokens* nastavena tak, aby zobrazovala verzi Nginx na automaticky generovaných chybových stránkách a v hlavičkách HTTP odpovědí. Aby tyto informace nebyly veřejně zobrazeny, mělo by toto chování být zakázáno nastavením *server_tokens off* v konfiguračním souboru Nginx.

5. Omezení velikosti vyrovnávací paměti klienta

Nastavení omezení velikosti vyrovnávací paměti klienta může pomoci chránit webový server před útoky přetečením vyrovnávací paměti a útoky DoS. Řídí se to následujícími směrnicemi v konfiguračním souboru:

- *client_body_buffer_size* slouží k určení velikosti vyrovnávací paměti těla požadavku klienta; výchozí hodnota je 8k nebo 16k, ale doporučuje se nastavit ji na 1k,
- *client_header_buffer_size* slouží k určení velikosti vyrovnávací paměti hlavičky požadavku klienta; pro většinu požadavků postačí velikost 1 kB,
- *client_max_body_size* určuje maximální akceptované velikosti těla pro požadavek klienta; velikost 1k je obvykle dostačující, ale pokud se používá metoda POST pro nahrání souborů uživatelem, je nutné ji zvýšit,

- *large_client_header_buffers* určuje maximální počet a velikost vyrovnávacích pamětí, které se mají použít ke čtení hlaviček velkých požadavků klientů.

4.6.3 SSL a TLS

Při výběru soukromých klíčů obvykle volba padá na 2 048bitové klíče RSA – a je to poměrně bezpečná sázka. Takové klíče poskytují 112bitové zabezpečení, dostačující pro většinu webových stránek, a jsou široce podporovány. Pro vyšší úroveň zabezpečení lze použít 3 072bitové klíče RSA, ale to ovlivní výkon, protože klíče RSA se špatně škálují. Klíče ECDSA mohou tento problém vyřešit, protože poskytují 128bitové zabezpečení již na 256 bitech. Pokud jde o ochranu samotných klíčů, měly by být chráněny heslem pro zajištění jejich ochrany v zálohovacích systémech. Certifikáty a soukromé klíče by se měly obnovovat jednou ročně, případně i častěji.

Pro většinu webových stránek bývá používání klíčů RSA silnějších než 2 048 bitů a klíčů ECDSA silnějších než 256 bitů plýtváním výkonem procesoru a může zhoršit UX. Neexistují žádné prokazatelné výhody používání šifrování nad 128 bitů.

Serverový SSL certifikát se používá k zajištění legitimity webhostingu. Webový server by měl mít certifikát fungující pro každý název DNS, který na něj odkazuje. Přinejmenším by konfigurace měla obsahovat adresy URL s příponou „www“ i bez ní. Zabezpečení certifikátu závisí na síle soukromého klíče, který byl použit k podpisu certifikátu, a na síle hashovací funkce použité v podpisu.

Certifikáty by měl dodávat spolehlivý zdroj, nazývaný certifikační autoritou (CA). Při výběru certifikační autority je třeba vzít v úvahu několik skutečností:

1) jak velký důraz kladou na zabezpečení – jak reagují na kompromitace, jestli existují nějaké problematické momenty v jejich historii zabezpečení,
2) hlavní oblast podnikání – zda jsou certifikáty hlavním zaměřením CA a jejich oblastí odbornosti,
3) nabízené služby – metody odvolání Certificate Revocation List (CRL) a Online Certificate Status Protocol (OCSP), certifikáty Extended Validation (EV), výběr algoritmů veřejného klíče, pokročilé nástroje pro správu certifikátů, podpora,

4) síťovou dostupnost a výkon CA.

Autorizace certifikační autority DNS (Certification Authority Authorization, CAA) [31] je standard umožňující vlastníkům doménových jmen omezit, které CA mohou vydávat certifikáty pro jejich domény. Díky zavedenému CAA je plocha pro útoky na podvodné certifikáty omezena, což zvyšuje zabezpečení stránek. Doporučuje se přidat důvěryhodné CA na bílou listinu přidáním záznamu CAA k certifikátu.

K vytvoření úplného řetězu důvěry jsou potřeba dva nebo více serverových certifikátů. Aby se předešlo problémům s nasazením způsobeným absencí zprostředkujících certifikátů, doporučuje se používat všechny certifikáty v poskytnutém CA pořadí. Neúplné řetězy certifikátů mohou mít za následek varování prohlížeče, protože certifikát hlavního serveru bude vnímán jako neplatný. Ne všechny prohlížeče jsou schopny rekonstruovat chybějící zprostředkující certifikáty.

V rodině SSL/TLS existuje šest protokolů: SSL v2, SSL v3, TLS v1.0, TLS v1.1, TLS v1.2 a TLS v1.3. Avšak pouze TLS v1.2 a v1.3 nemají žádné známé bezpečnostní problémy, navíc poskytují lepší výkon. Zbývající protokoly jsou zastaralé: nejsou podporovány moderními prohlížeči a nejsou povoleny v souladu s PCI [32].

Bezpečná internetová komunikace funguje za předpokladu, že komunikace probíhá přímo s požadovanou stranou. V SSL a TLS šifrovací sady definují, jak probíhá bezpečná komunikace. Skládají se z různých stavebních bloků s cílem dosažení bezpečnosti prostřednictvím rozmanitosti. Pokud se zjistí, že jeden ze stavebních bloků je slabý nebo nedůvěryhodný, je možné přejít na jiný. Doporučuje se dávat přednost následujícím šifrovacím sadám:

- AEAD (Authenticated Encryption with Associated Data) šifrovací sady – CHACHA20_POLY1305, GCM a CCM,
- PFS (Perfect Forward Secrecy) šifry – ECDHE_RSA, ECDHE_ECDSA, DHE_RSA, DHE_DSS, CECPQ1, všechny šifry TLS 1.3.

Ve verzích protokolu SSL v1.3 a novějších klienti odesílají seznam šifrovacích sad, které podporují, a servery si ze seznamu vybírají jednu sadu pro připojení. Pro

adekvátní ochranu serveru je nezbytné vybírat nejlepší dostupnou šifrovací sadu, a ne pouze první v seznamu.

Forward secrecy je funkce protokolu, která umožňuje zabezpečené konverzace, jež nejsou závislé na soukromém klíči serveru. Se šifrovacími sadami, které neposkytují dopředné utajení, může někdo, kdo dokáže obnovit soukromý klíč serveru, dešifrovat všechny dříve zaznamenané šifrované konverzace. Sady ECDHE musí být upřednostňovány, aby tato funkce byla umožněna s moderními webovými prohlížeči. V případě, že je potřeba podporovat širší škálu klientů, mohou být jako záložní použity sady DHE. V tomto ohledu se nedoporučuje používat klíče RSA.

Pro výměnu klíčů si veřejné stránky mohou obvykle vybrat mezi klasickou efemérní výměnou klíčů Diffie-Hellman (DHE) a její variantou eliptické křivky, ECDHE. Některé zranitelnosti byly objeveny [33] pro DH s nižší silou (1 024 DH a méně). Proto je 2 048 DH nejlepší volba. Výměna klíčů ECDHE se však ukazuje jako lepší z hlediska výkonu, a proto může být zvolena namísto ní. Zvýšení síly efemérní výměny klíčů nad 2 048 bitů pro DHE a 256 bitů pro ECDHE má jen malý přínos, zato špatný vliv na výkon.

Výkon, a tím pádem i škálovatelnost klíčů mají pro digitální agenturu velkou důležitost, proto by nejvhodnější volbou mohla být kombinace RSA a ECDSA klíčů, co se týká autentizace. Produkty firmy do určité míry podporují starší prohlížeče, a pro výměnu klíčů jsou zapotřebí DHE sady, avšak ve značně omezené míře. V celku se doporučuje využívat PFS šifrovací sady.

5 Shrnutí a závěr

Servery obsahují velké množství citlivých informací i dalších důležitých aktiv firmy. Z tohoto důvodu představují atraktivní cíl pro kyberzločince, často za účelem finančního zisku. Výsledky úspěšného útoku mohou být pro podniky devastující. Údržba zabezpečení serveru je tak nedílnou součástí podnikání.

Cílem bakalářské práce bylo připravit podklad pro rozhodování o implementaci zabezpečení webového serveru pro konkrétní podnik. Řešení bylo navrženo na základě informací o službách nabízených různými servery, o výsledcích průzkumů trhu a znalostech nejběžnějších typů útoků, pojednávaných v teoretické části práce. Dále na výsledné řešení měly vliv požadavky firmy, uvedené v praktické části. V následujícím odstavci bude představeno shrnutí výsledků, k nimž se v práci dospělo.

Fyzickou stránkou zabezpečení se firma nebude zabývat a využije hostingových služeb. Stávající řešení zálohování je z bezpečnostního hlediska vyhovující. Z důvodu reprezentace na trhu a kompatibility serverů Apache a Nginx budou zvoleny oba tyto servery. Konkrétně Apache jako backendový server a Nginx jakožto proxy server. Oba servery budou muset být vhodně zabezpečeny, aby byly v souladu s mezinárodními standardy a co nejlépe bránily úniku dat. Kromě proxy serveru bude pro komplexnější zabezpečení použit firewall síťové vrstvy. Doporučeny jsou protokoly TLS v1.2 a TLS v1.3 a šifrovací sady s Perfect Forward Secrecy.

6 Seznam použité literatury

[1] IBM, „Cost of a Data Breach Report 2020". IBM, 2020. [Online]. Dostupné z: https://www.ibm.com/security

[2] Verizon, „2021 Data Breach Investigations Report". Verizon, 2021. [Online]. Dostupné z: https://www.verizon.com/business/resources/reports/2021-data-breach-investigations-report.pdf

[3] „What is Web Server Architecture?", *Techopedia*. [Online]. Dostupné z: http://www.techopedia.com/definition/30263/web-server-architecture

[4] Oracle Corporation, „Oracle iPlanet Web Server 7.0.9 Developer's Guide". Oracle Corporation, 2010. [Online]. Dostupné z: https://docs.oracle.com/cd/E19146-01/821-1829/index.html

[5] R. Fielding a J. Reschke, „Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC Editor, RFC7230, čer. 2014. doi: 10.17487/rfc7230.

[6] A. Sunyaev, *Internet Computing: Principles of Distributed Systems and Emerging Internet-Based Technologies*. Cham: Springer International Publishing, 2020. doi: 10.1007/978-3-030-34957-8.

[7] „What is a web server?" MDN. [Online]. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn/Common_questions/What_is_a_web_server

[8] „October 2021 Web Server Survey". Netcraft, 15. říjen 2021. [Online]. Dostupné z: https://news.netcraft.com/archives/2021/10/15/october-2021-web-server-survey.html

[9] R. Bowen a K. Coar, *Apache Cookbook: Solutions and Examples for Apache Administration*. O'Reilly Media, Inc., 2007.

[10] D. Kegel, „The C10K problem". 5. únor 2014. [Online]. Dostupné z: http://www.kegel.com/c10k.html

[11] D. DeJonghe, *NGINX Cookbook*. O'Reilly Media, Inc., 2020.

[12] „Documentation for NGINX Amplify". NGINX. [Online]. Dostupné z: https://amplify.nginx.com/docs/

[13] F. Memon, „A Guide to Caching with NGINX and NGINX Plus", *NGINX Tech Blog*, 23. červenec 2015. https://www.nginx.com/blog/nginx-caching-guide

[14] „Restricting Access with HTTP Basic Authentication". NGINX. [Online]. Dostupné z: https://docs.nginx.com/nginx/admin-guide/security-controls/configuring-http-basic-authentication

[15] S. Peyrott, „JWT Handbook". Auth0, 2018. [Online]. Dostupné z: https://auth0.com/resources/ebooks/jwt-handbook

[16] P. Jirásek, L. Novák, a J. Požár, „Výkladový slovník kybernetické bezpečnosti". Národní úřad pro kybernetickou a informační bezpečnost, 2015. [Online]. Dostupné z: https://www.nukib.cz/cs/kyberneticka-bezpecnost/regulace-a-kontrola/podpurne-materialy/

[17] „What is an IT Asset?", *Techopedia*. [Online]. Dostupné z: http://www.techopedia.com/definition/16946/it-asset

[18] U. H. Rao a U. Nayak, „The InfoSec Handbook", in *The InfoSec Handbook: An Introduction to Information Security*, U. H. Rao a U. Nayak, Ed. Berkeley, CA: Apress, 2014, s. 115–139. doi: 10.1007/978-1-4302-6383-8_6.

[19] „OWASP Top Ten Web Application Security Risks". OWASP, 2021. [Online]. Dostupné z: https://owasp.org/www-project-top-ten/

[20] O. Tayan, „Concepts and Tools for Protecting Sensitive Data in the IT Industry: A Review of Trends, Challenges and Mechanisms for Data-Protection", *Int. J. Adv. Comput. Sci. Appl.*, roč. 8, č. 2, 2017, doi: 10.14569/IJACSA.2017.080207.
[21] M. Johns, „Session Hijacking Attacks", in *Encyclopedia of Cryptography and Security*, H. C. A. van Tilborg a S. Jajodia, Ed. Boston, MA: Springer US, 2011, s. 1189–1190. doi: 10.1007/978-1-4419-5906-5_661.
[22] K. Daimi a C. Peoples, Ed., *Advances in Cybersecurity Management*. Cham: Springer International Publishing, 2021. doi: 10.1007/978-3-030-71381-2.
[23] A. Hoffman, *Web Application Security Exploitation and Countermeasures for Modern Web Applications*. O'Reilly Media, 2020.
[24] PCI Security Standards Council, „PCI DSS Quick Reference Guide". PCI Security Standards Council, červenec 2018. [Online]. Dostupné z: https://www.pcisecuritystandards.org/document_library
[25] ISO, „ISO/IEC 27001:2013". ISO, říjen 2013. [Online]. Dostupné z: https://www.iso.org/standard/54534.html
[26] World Wide Web Technology Surveys, „Usage statistics of web servers". 2022. [Online]. Dostupné z: https://w3techs.com/technologies/overview/web_server
[27] K. Schroeder, „Performance of Apache 2.4 with the event MPM compared to Nginx", *ESchrade – Kevin Schroeder*, 3. leden 2014. https://www.eschrade.com/page/performance-of-apache-2-4-with-the-event-mpm-compared-to-nginx/
[28] T. Butler, „Apache vs Nginx vs OpenLiteSpeed: Part 1". Conetix, 26. březen 2015. [Online]. Dostupné z: https://conetix.com.au/blog/apache-vs-nginx-vs-openlitespeed-part-1/
[29] NGINX, „Like Apache: .htaccess". NGINX. [Online]. Dostupné z: https://www.nginx.com/resources/wiki/start/topics/examples/likeapache-htaccess/
[30] A. Brown, Ed., *Structure, scale and a few more fearless hacks*. Mountain View: Creative Commons, 2012.
[31] P. Hallam-Baker a R. Stradling, „DNS Certification Authority Authorization (CAA) Resource Record", Internet Engineering Task Force, Request for Comments RFC 6844, led. 2013. doi: 10.17487/RFC6844.
[32] K. A. McKay a D. A. Cooper, „Guidelines for the selection, configuration, and use of Transport Layer Security (TLS) implementations", National Institute of Standards and Technology, Gaithersburg, MD, NIST SP 800-52r2, srp. 2019. doi: 10.6028/NIST.SP.800-52r2.
[33] D. Adrian *et al.*, „Imperfect Forward Secrecy: How Diffie-Hellman Fails in Practice", in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, Denver Colorado USA, říj. 2015, s. 5–17. doi: 10.1145/2810103.2813707.

Univerzita Hradec Králové
Fakulta informatiky a managementu

Zadání bakalářské práce

Autor: **Irina Fogelzang**

Studium: I1800166

Studijní program: B1802 Aplikovaná informatika

Studijní obor: Aplikovaná informatika

Název bakalářské práce: **Zabezpečení webového serveru proti odcizení dat**

Název bakalářské práce AJ: Web server protection against data breaches

Cíl, metody, literatura, předpoklady:

Cílem práce je poskytnout přehled technik používaných k zabezpečení webového serveru před odcizením dat. Dále uvést způsoby, jak k úniku dat může dojít (od chyb při konfiguraci a správě serveru po konrétní typy útoků), popsat obecné směry zabezpečení (interní a externí hrozby), s důrazem na specifika webových serverů a technik jejich zabezpečení. V praktické části student navrhne plán pro zabezpečení serveru pro potřeby konkrétní firmy.

1. MCCLURE, Stuart; SCAMBRAY, Joel; KURTZ, George. Hacking Exposed 5th Edition (Hacking Exposed). 2005.
2. RAO, Umesh Hodeghatta; NAYAK, Umesha. *The InfoSec handbook: An introduction to information security*
   . Apress, 2014.
3. *TILBORG, Henk CA van; JAJODIA, Sushil. Encyclopedia of Cryptography and Security, 2011.*
4. *HOFFMAN, Andrew. Web Application Security: Exploitation and Countermeasures for Modern Web Applications*
   . O'Reilly Media, 2020.
5. *DEJONGHE, Derek. Nginx CookBook*
   . O'Reilly media, 2021

Garantující pracoviště: Katedra informačních technologií,
Fakulta informatiky a managementu

Vedoucí práce: doc. Ing. Vladimír Soběslav, Ph.D.

Datum zadání závěrečné práce: 28.5.2021