UNIVERZITA PARDUBICE

FAKULTA ELEKTROTECHNIKY A INFORMATIKY

DIPLOMOVÁ PRÁCE

2020 Bc. Martin Volenec

Univerzita Pardubice

Fakulta elektrotechniky a informatiky

Bezpečnost webových aplikací

Bc. Martin Volenec

Diplomová práce

2020

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: 2019/2020

ZADÁNÍ DIPLOMOVÉ PRÁCE
(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Martin Volenec**
Osobní číslo:	**I18251**
Studijní program:	**N2646 Informační technologie**
Studijní obor:	**Informační technologie**
Téma práce:	**Bezpečnost webových aplikací**
Zadávající katedra:	**Katedra softwarových technologií**

Zásady pro vypracování

Cílem diplomové práce je provést analýzu a rešerši problematiky bezpečnosti webových aplikací. V praktické části budou vytvořeny ukázkové aplikace řešící vybrané typy problémů. V teoretické části bude provedena rešerše a analýza problematiky autentizace a autorizace u webových aplikací – použitelné metody (u aplikací a webových služeb), bezpečnost hesla, obnovení zapomenutého hesla, jednorázová hesla, externí autentizační mechanismy a problematika více faktorové autentizace. Dále bude popsána problematiky útoků na webové stránky (např.: CSS/XSS, CSRF) a možnosti prevence; standard security.txt; bezpečnostní prvky související s protokoly HTTP – CSP, CORS; bezpečnost HTTPS protokolu a jeho konfigurace; bezpečnost související s nastavením webového serveru a operačního systému. V praktické části budou vytvořeny demonstrační aplikace, řešící vybrané problémy (např. autentizace – jménem a heslem, pomocí cookie, pomocí JWT, ...). Demonstrační aplikace budou realizovány v jazyce PHP s využitím moderních webových frameworků.

Rozsah pracovní zprávy: **50-60 stran**
Rozsah grafických prací:
Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam doporučené literatury:

WU, Hanqing a Liz ZHAO. Web Security: A WhiteHat Perspective. CRC Press, 2015. ISBN 9781466592629.
SNYDER, Chris a Michael SOUTHWELL. Pro PHP Security. Apress, 2006. ISBN 9781430200574.

Vedoucí diplomové práce: **Ing. Roman Diviš**
Katedra softwarových technologií

Datum zadání diplomové práce: **5. listopadu 2019**
Termín odevzdání diplomové práce: **15. května 2020**

Univerzita Pardubice Česká republika
L.S.

Ing. Zdeněk Němec, Ph.D.
děkan

prof. Ing. Antonín Kavička, Ph.D.
vedoucí katedry

V Pardubicích dne 15. listopadu 2019

Prohlášení autora

Prohlašuji, že jsem tuto práci vykonal samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména ze skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně.

V Pardubicích dne 15. 05. 2020

Bc. Martin Volenec

PODĚKOVÁNÍ

Zde bych chtěl poděkovat panu Ing. Romanu Divišovi za strávený čas a cenné rady při zpracování diplomové práce. Hlavně za rychlé odezvy při dotazech a revizích. Dále bych chtěl poděkovat všem blízkým za poskytnutou oporu během celého studia.

ANOTACE

Práce se zaměřuje na rozbor vybraných témat spojených s moderní kryptografií, bezpečností webových aplikací a síťového stacku. Taktéž disponuje ukázkovými kódy a jednoduchými aplikacemi v jazyce PHP, které podrobně demonstrují určitou problematiku zabezpečení a navrhují její řešení.

KLÍČOVÁ SLOVA

Kyberbezpečnost, kryptografie, autentizace, autorizace, PHP, OWASP, HTTPS

TITLE

Web application security

ANNOTATION

The thesis focuses on the analysis of selected topics related to modern cryptography, security of web application and network stack. Available are sample codes and basic applications developed in PHP. They are demonstrating in detail problems of security and proposing their solution.

KEYWORDS

Cybersecurity, cryptography, authentication, authorization, PHP, OWASP, HTTPS

OBSAH

SEZNAM ILUSTRACÍ A TABULEK

SEZNAM ZDROJOVÝCH KÓDŮ

SEZNAM ZKRATEK A ZNAČEK

AES	Advanced Encryption Standard
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
CSP	Content-Security-Policy
CSRF	Cross-site Request Forgery
DBMS	Database Management System
DOM	Document Object Model
HMAC	Keyed-hash Message Authentication Code
HSTS	HTTP Strict Transport Security
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
ICV	Integrity Check Value
JSON	JavaScript Object Notation
JWT	JSON Web Tokens
LDAP	Lightweight Directory Access Protocol
MAC	Message Authentication Code
MIC	Message Integrity Check
MIME	Multipurpose Internet Mail Extensions
MITM	Man in the middle
OWASP	Open Web Application Security Project
PHP	Hypertext Preprocessor
PIN	Personal Identification Number
RFC	Request for Comments
RSA	Rivest, Shamir, Adleman
SCT	Signed Certificate Timestamp
SEO	Search Engine Optimization
SFTP	Secure File Transfer Protocol
SQL	Structured Query Language
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol

TLS	Transport Layer Security
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
UTC	Coordinated Universal Time
XML	Extensible Markup Language
XSS	Cross-site scripting

ÚVOD

Nacházíme se na vrcholu digitální revoluce a před námi stojí čtvrtá průmyslová revoluce – průmysl 4.0. Ta přinese do našeho každodenního života spoustu nových a pokročilých technologií, které změní náš život. Ať už mluvíme o síti 5G, internetu věcí, pokročilé umělé inteligenci či kvantových počítačích. Tyto technologie budou čím dál větší součástí našeho života. Avšak s každou novou technologií přichází taktéž odpovědnost, výzva a hrozba. A tomu všemu je potřeba se postavit čelem.

Velká část našeho světa či života je již na internetu. Pracovní život, soukromý život na sociálních sítích či naše nové chytré auto. Všechny digitální součásti našeho života čelí stejnému nebezpečí – kybernetické útoky. Ty jsou bohužel zcela běžnou součástí digitalizace a vždy jim budeme muset čelit.

Dnešní povědomost o kyberbezpečnosti v již pokročilém digitálním věku je stále neuspokojivá. A to ať už mezi běžnými uživateli, kde se to dá očekávat a částečně tolerovat, či například mezi vývojáři. Tato skutečnost mě inspirovala k vytvoření práce, ve které bych rád demonstroval své znalosti o bezpečnosti nejen webových aplikací. Další motivací je snaha zvýšení obecné povědomosti o této problematice.

Je potřeba si uvědomit, že bezpečnost či zabezpečení není jedno ohraničené téma. Skládá se z mnoha dílčích témat, která do sebe zapadají a navzájem se ovlivňují. A jako celek tvoří téma bezpečnost. Tato práce popisuje jen vybrané části, neboť popsat všechny bezpečnostní hrozby je dnes prakticky nemožné.

Diplomová práce se zabývá rozborem vybraných témat bezpečnosti v prostředí webových aplikací. Postupně jsou probírána témata moderní kryptografie, autentizace a autorizace, bezpečnost webové aplikace a bezpečnost síťového stacku.

Každý vývojář by měl disponovat alespoň základními znalostmi kyberbezpečnosti. Pevně věřím, že přečtením této práce čtenář získá přehled a hodnotné informace v prostředí bezpečnosti webových aplikací.

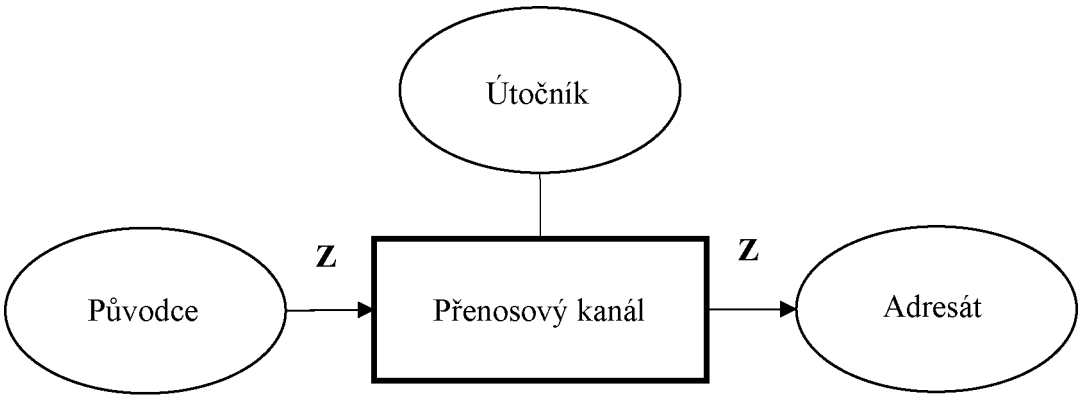
1 ÚVOD DO MODERNÍ KRYPTOGRAFIE

Pokud chceme ochránit data v počítači či přes internet operovat se svým bankovním účtem, tak jsou data šifrována. Pokud chceme mít záruku, že elektronické dokumenty jsou autentické, tak je potřeba je digitálně podepisovat. To všechno, a mnohé další, jsou příklady využití kryptografie v praxi. Kryptografii lze definovat jako vědu, která se zabývá konstrukcí a aplikací matematických metod k zajišťování důvěrnosti a autentičnosti zpráv [3].

Obrázek 1 zobrazuje problematiku bezpečnosti zpráv. Zprávu **Z** můžeme interpretovat jako posloupnost symbolů se zakódovanou informací. A to v podobě textu, obrázku či telefonních hovorů. Pomocí digitalizace lze zprávu převést na posloupnost čísel. Tato číselná podoba lze poté zabezpečit pomocí matematických operací.

Původce je zdroj zprávy a adresát je její zamýšlený příjemce. Souhrnně je lze nazývat oprávněné osoby. Zprávy jsou přenášeny prostřednictvím přenosového kanálu, pod kterým si můžeme představit například Wi-Fi spojení.

Nicméně k přenosovému kanálu má přístup taktéž útočník. Útočník je neoprávněná osoba, která může zprávy buď odposlouchávat (tzv. pasivní útok) nebo je modifikovat (tzv. aktivní útok). S těmito útoky souvisí důvěrnost a autentičnost zpráv. Důvěrnost je stav, kdy obsah zprávy je znám pouze původci a adresátovi [3]. Autentičnost je stav, kdy předpoklady o původu přijaté zprávy jsou pravdivé [3]. Adresát si může ověřit, že zpráva skutečně pochází od předpokládaného původce.



Obrázek 1 – Obecné schéma přenosu zpráv. Zdroj: [3]

Kryptoanalýza je komplementární vědou ke kryptografii. Tu lze analogicky definovat jako vědu o překonávání či lámání matematických metod, které zajišťují důvěrnost a autentičnost zpráv [3]. Kryptografie a kryptoanalýza společně tvoří vědu kryptologie.

Další vědou, která se z kryptografie často vyčleňuje, je steganografie [4]. Smyslem této vědy je ukrytí zprávy ve smyslu její samotné existence. Zpráva putuje komunikačním kanálem takovým způsobem, aby byla nenápadná či neodhalena.

Kryptografické techniky jsou dnes již automatickou součástí moderních komunikačních a informačních systémů. Díky nim lze konstruovat řadu systému a služeb, jako jsou elektronické platební systémy, elektronické měny či volby.

1.1 Kryptografické systémy

Abychom zajistili důvěrnost zprávy **Z**, tak ji musíme transformovat na číselnou posloupnost **C**. Tato číselná posloupnost se nazývá kryptogram. Platí, že ke kryptogramu má přístup kdokoliv. Důvěrnost zprávy je zajištěna tím, že k provedení transformace kryptogramu zpět na zprávu je zapotřebí tajná informace **K** (tzv. klíč).

V případě zajištění autentičnosti zprávy, je zpráva **Z** rozšířena o číselnou posloupnost **P** (tzv. pečeť). Tato posloupnost čísel závisí na původní zprávě **Z** a na tajném klíči **K**, který zná jen původce. Adresát poté správnost pečeti kontroluje a ověří si autentičnost přijaté zprávy.

Pod pojmem kryptografický systém rozumíme algoritmus určený k zajištění důvěrnosti a/nebo autentičnosti zpráv [3].

1.1.1 Utajovací kryptografické systémy

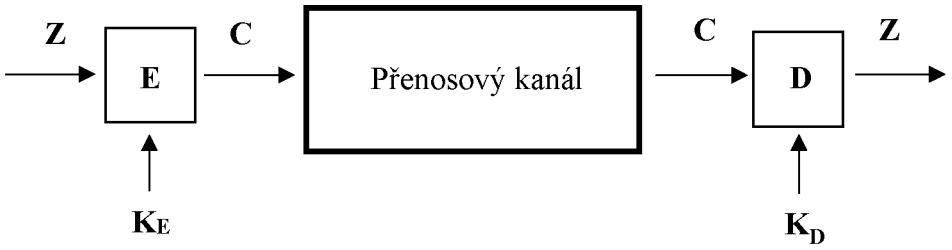
Utajovací kryptografické systémy zajišťují důvěrnost zpráv [3]. Obrázek 2 zobrazuje fungovaní systému. Pro korektní fungování je potřeba dvou parametrů. Na straně původce je to číslo $\mathbf{K_E}$ a nazývá se šifrovací klíč. Na straně adresáta potřebujeme číslo $\mathbf{K_D}$, které se nazývá dešifrovací klíč.

E a **D** jsou speciální matematické funkce (transformace), které se nazývají šifrování („encrypting“) a dešifrování („decrypting“). Vstupem šifrovací funkce **E** je šifrovací klíč $\mathbf{K_E}$ a zpráva **Z**. Výstupem je číselná posloupnost, tedy kryptogram **C**. Tento proces popisuje vzorec $\boldsymbol{C = E\,(Z, K_E)}$ [3].

Šifrovací funkce musí být zkonstruována tak, aby útočník s předpokládanou úrovní možností nebyl schopen z daného kryptogramu bez znalosti dešifrovacího klíče odvodit zprávu dříve, než je stanovená doba rezistence kryptogramu. Tato doba je individuální a závisí na potřebách dané aplikace [3].

Na straně adresáta se provádí dešifrování pomocí funkce **D**. Jedná se o inverzní funkci k šifrování. Dešifrovací funkce lze vyjádřit následovně: $\boldsymbol{D}\ (\boldsymbol{C}, \boldsymbol{K_D}) = \boldsymbol{Z}$ [3].

Mezi šifrovacím a dešifrovacím klíčem existuje závislost: $\boldsymbol{K_D} = \boldsymbol{f}(\boldsymbol{K_E})$ [3].



Obrázek 2 – Obecné schéma utajovacího kryptografického systému. Zdroj: [3]

Pokud je prakticky možné, v průběhu doby rezistence kryptogramu, zjistit hodnotu dešifrovacího klíče $\mathbf{K_D}$ z hodnoty šifrovacího klíče $\mathbf{K_E}$, tak oba klíče musí být tajné. Jelikož je utajení obou klíčů stejné, nacházejí se v tzv. symetrickém postavení. Proto se tyto kryptografické systémy nazývají symetrické utajovací kryptografické systémy. Do této třídy můžeme zařadit například blokovou šifru AES [3].

Avšak pokud je prakticky nemožné, v průběhu doby rezistence programu, zjistit hodnotu dešifrovacího klíče $\mathbf{K_D}$ z hodnoty šifrovacího klíče $\mathbf{K_E}$, tak šifrovací klíč může být veřejně známý (tzv. veřejný klíč) a utajovat je zapotřebí pouze klíč dešifrovací (tzv. soukromý klíč). Jelikož je utajení obou klíčů rozdílné, nacházejí se v tzv. asymetrickém postavení. Tudíž se tyto kryptografické systémy nazývají asymetrické utajovací kryptografické systémy. Příkladem může být šifra RSA [3].

1.1.2 Autentizační kryptografické systémy

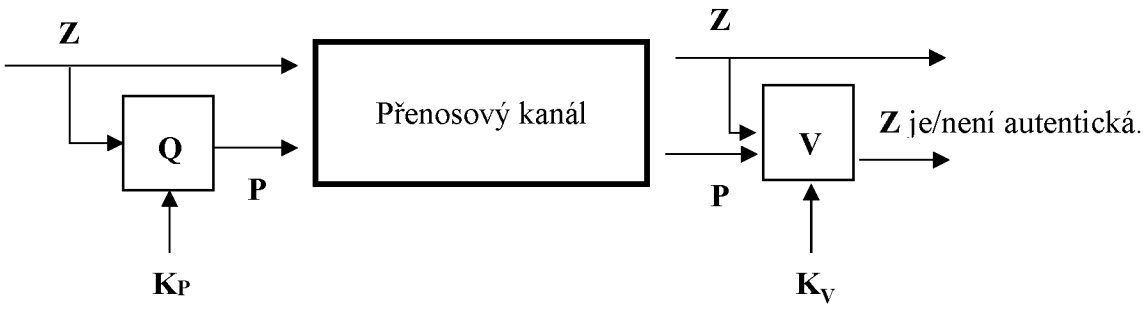
Autentizaci zpráv zajišťují autentizační kryptografické systémy [3]. Obrázek 3 zobrazuje fungování systému. Opět je zapotřebí dvou parametrů. Na straně původce je to číslo **K**P a nazývá se pečetící klíč. Na straně adresáta je zapotřebí číslo **K**V, které se nazývá ověřovací klíč.

Q a **V** jsou speciální matematické funkce (transformace), které se nazývají pečetění a ověřování. Vstupem pečetící funkce **Q** je pečetící klíč **K**P, která každé zprávě **Z** přiřadí číslo **P** (tzv. pečeť). V praxi se toto číslo nazývá různě — například MAC, MIC, ICV, digitální podpis („Digital signature“) atd. Tvorbu pečeti lze vyjádřit následovně: $\boldsymbol{P = Q\,(Z, K_P)}$ [3].

Pečetící funkce musí být zkonstruována tak, aby útočník s předpokládanou úrovní možností nebyl schopen bez znalosti pečetícího klíče odvodit k libovolné zprávě správnou pečeť dříve, než je stanovená doba rezistence pečeti.

Adresát (ověřovatel) provede ověření funkcí **V**. To je taková funkce, jejímž vstupem je zpráva **Z**, pečeť **P** a ověřovací klíč **K**V. Výsledkem funkce je, že zpráva **Z** je buď autentická, nebo není autentická.

Opět existuje mezi pečetícím a ověřovacím klíčem závislost: $\boldsymbol{K_P = f(K_V)}$ [3].



Obrázek 3 – Obecné schéma autentizačního kryptografického systému. Zdroj: [3]

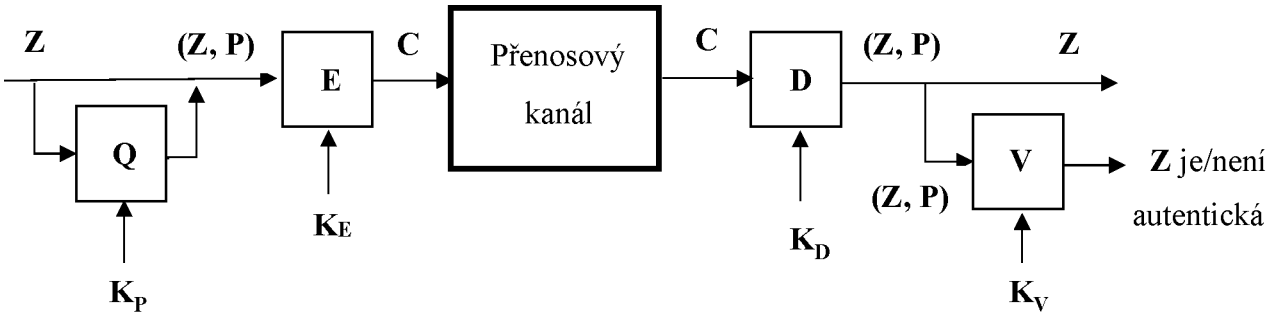
Pokud je prakticky možné, v průběhu doby rezistence pečeti, zjistit pečetící klíč $\mathbf{K_P}$ z hodnoty ověřovacího klíče $\mathbf{K_V}$, tak oba klíče musí být tajné. Jelikož je utajení obou klíčů stejné, nacházejí se v tzv. symetrickém postavení. Proto se tyto kryptografické systémy nazývají symetrické autentizační kryptografické systémy. Jako reprezentanta této třídy lze uvést techniku HMAC [3].

Avšak pokud je prakticky nemožné, v průběhu doby rezistence programu, zjistit hodnotu klíče $\mathbf{K_P}$ z hodnoty klíče $\mathbf{K_V}$, tak ověřovací klíč může být veřejně známý (tzv. veřejný klíč) a utajovat je zapotřebí pouze pečetící klíč (tzv. soukromý klíč). Protože je utajení obou klíčů nestejné, nacházejí se v tzv. asymetrickém postavení. Proto se tyto kryptografické systémy nazývají asymetrické autentizační kryptografické systémy. Příkladem mohou být systémy digitálního podpisu [3].

1.1.3 Hybridní kryptografické systémy

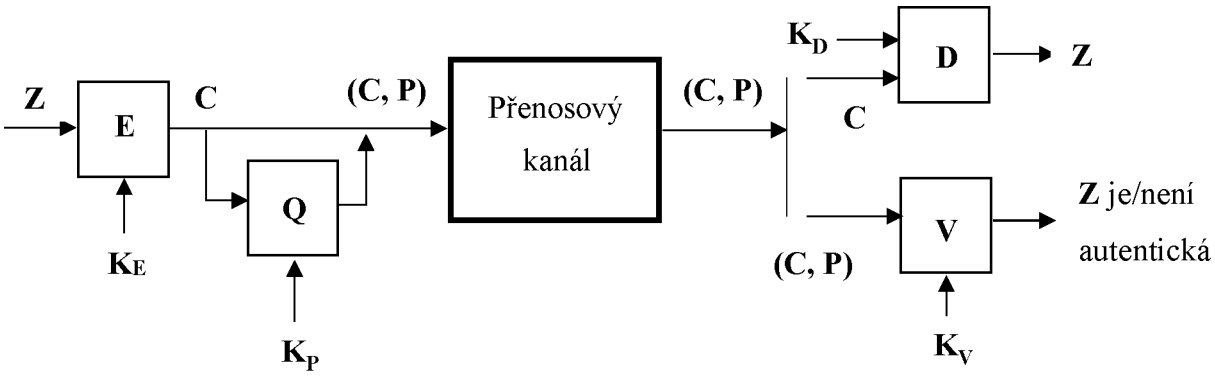
Pokud je potřeba zajistit důvěrnost a autentičnost zpráv současně, tak se buď zašifruje zapečetěná zpráva, nebo se zapečetí zašifrovaná zpráva [3].

Obrázek 4 zobrazuje schéma hybridního kryptografického systému, kde se šifrují zapečetěné zprávy. Nejdříve je zpráva **Z** opatřena pečetí **P**, poté šifrovací funkcí **E** převedena na kryptogram **C**. Adresát kryptogram **C** dešifruje a následně si ověří autentičnost zprávy za pomoci její pečeti **P**.



Obrázek 4 – Hybridní kryptografický systém, kde se šifruje zapečetěná zpráva. Zdroj: [3]

Druhá je možnost je hybridní kryptografický systém, kde se pečetí zašifrovaná zpráva. Tu zobrazuje obrázek 5. Zpráva **Z** je nejprve zašifrována funkcí **E** a pak se zapečetí kryptogram **C**. Adresát nejprve ověří autentičnost pečeti **P** a poté zprávu dešifruje funkcí **D**.



Obrázek 5 – Hybridní kryptografický systém, kde se pečetí zašifrovaná zpráva. Zdroj: [3]

1.2 Jednosměrné funkce

Jednosměrná funkce f je funkce, pro jejíž libovolný vzor x lze snadno vypočítat obraz $y = f(x)$, avšak pro daný obraz y je prakticky nemožné vypočítat jeho vzor x. V současné době není dokázáno, zda takováto třída funkcí vůbec existuje. V praxi se používají funkce, u nichž se věří, že popsanou vlastnost mají [3].

Jednosměrné funkce lze klasifikovat na:

- Funkce s pevnou délkou výstupu.
- Funkce s volitelnou délkou výstupu.

Jednosměrné funkce s pevnou délkou výstupu přiřadí vzoru o libovolné bitové délce určitý obraz, který má pěvně stanovenou délku. Tato funkce je obvykle nazývána hešovací („hash function“). Slouží k přiřazení reprezentantu zprávě **Z**.

Funkce s volitelnou délkou vstupu lze dále dělit na:

- Kompresní funkce — výstup kratší než vstup.
- Ekvivalentní funkce — výstup stejný jako vstup.
- Expanzní funkce — výstup delší než vstup.

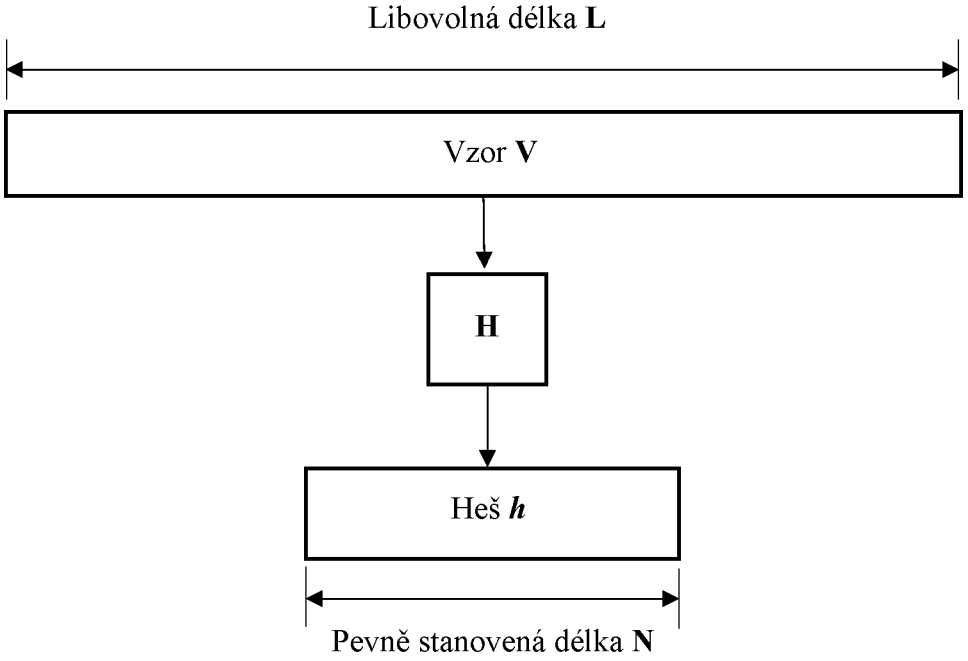
1.2.1 Hešovací funkce

Účelem této funkce je vytvářet reprezentanty zpráv o konstantní délce (tzv. heše) [3].

Vstupem hešovací funkce **H** je binární posloupnost (zpráva) o libovolné délce **L** bitů. Výstupem funkce je obraz ***h*** (tzv. heš) o pevně stanovené délce **N**. V praxi typicky 128 až 512 bitů.

Hešovací funkce mají následující požadavky:

- Odolnost vůči získání vzoru.
- Odolnost vůči modifikaci vzoru.
- Odolnost vůči kolizím.



Obrázek 6 – Obecné schéma hešovací funkce. Zdroj: [3]

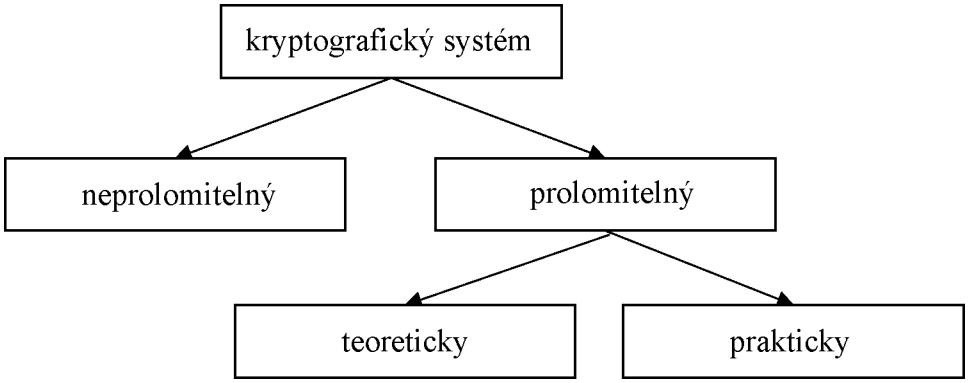
1.3 Bezpečnost kryptografických systémů

Bezpečnost je potřeba chápat jako odolnost vůči kryptoanalýze. Kryptografické systémy můžeme dělit z hlediska odolnosti na:

- neprolomitelné,
- prolomitelné.

V rámci neprolomitelných kryptografických systémů může mít útočník k dispozici neomezené výpočetní kapacity, a přesto nebude schopen systém prolomit. Nicméně neprolomitelné kryptografické systémy se kvůli svým provozním nevýhodám nepoužívají [3].

Prolomitelné jsou takové, kdy útočník bude mít potřebnou výpočetní kapacitu a systém prolomí. Pokud je však potřebná výpočetní kapacita mimo možnosti útočníka, tak je systém považován za teoreticky prolomitelný (tzv. výpočetně bezpečný). Zbylé jsou označovány za prakticky prolomitelné.



Obrázek 7 – Klasifikace kryptografických systému dle odolnosti. Zdroj: [3]

Moderní požadavky na kryptografické systémy a jejich bezpečnost zformuloval již v roce 1883 nizozemský lingvista a kryptolog Auguste Kerckhoffs. Publikoval celkem šest požadavků, přičemž čtyři z nich jsou již překonány technologickým vývojem [3].

Následující dva jsou stále platné:

- Vyzrazení kryptografického systému by nemělo jeho uživatelům způsobit problémy.
- Pokud kryptografický systém není neprolomitelný, měl by být prolomitelný pouze teoreticky.

První požadavek říká, že vyzrazení používaného kryptografického systému nesmí způsobit jeho uživatelům škody či problémy. Je zde vidět, že v tomto případě spočívá celkové zabezpečení pouze v neznalosti klíče útočníkem. Některé státní instituce a firmy se drží přístupu utajování kryptografického systému („security through obscurity") [3]. Slibují si, že útočník navíc musí získat popis používaného kryptografického systému. Nevýhodou tohoto přístupu je, že kryptografický systém neprošel veřejným hodnocením. Může tedy obsahovat slabiny, kterých si správci nemusí všimnout.

Druhý požadavek popisuje koncept výpočetní složitosti („computational security“). Tento koncept je v dnešní moderní kryptografii dominantní [3]. Říká, aby nové kryptografické systémy byly odolné vůči známým útokům, které budou útočníci vést s omezenými výpočetními možnostmi.

Je taktéž potřeba si uvědomit, že kryptologie je nekonečný souboj mezi kryptografií a kryptoanalýzou. S neustálým vývojem lidského poznání a technologických možností se stále objevují nové typy útoků, které tvůrci kryptografických systémů neznali a nemohli s nimi při návrhu počítat. Jedinou obranou vůči této skutečnosti je průběžné hodnocení bezpečnosti nasazených kryptografických systémů.

2 AUTENTIZACE A AUTORIZACE

Procesem autentizace a autorizace procházíme každodenně několikrát, aniž bychom si to vůbec uvědomovali. Ať už se jedná o odemčení mobilu, potvrzení bankovních plateb v účetnictví, či výběr peněz z bankomatu pomocí kódu PIN. Díky postupné digitalizaci našeho života i okolí, nabývají tyto procesy stále většího významu.

2.1 Autentizace

Autentizace je proces ověření identity subjektu. Jedná se o bezpečnostní opatření a ochranu před falšováním identity. V informatice se nejčastěji jedná o ověřování osob, zařízení nebo zpráv.

2.2 Autorizace

Po úspěšné autentizaci často následuje, i když to není podmínkou, proces autorizace. Jedná se o další bezpečnostní mechanismus, který zajišťuje kontrolu oprávnění k provedení určité operace. Už víme, kdo uživatel je, a nyní se ujišťujeme, že je k akci oprávněný.

S autorizací v informatice se často setkáme v podobě řízení přístupu k souborům a adresářům v operačních systémech. Dále pak při provádění operací, které může vykonat jen určitý uživatel (např. admin ve webové aplikaci).

2.3 Interní autentizační mechanismy

Interní autentizační mechanismy jsou takové mechanismy, které jsou součástí samotné aplikace a nevyužívají externí služby třetích stran.

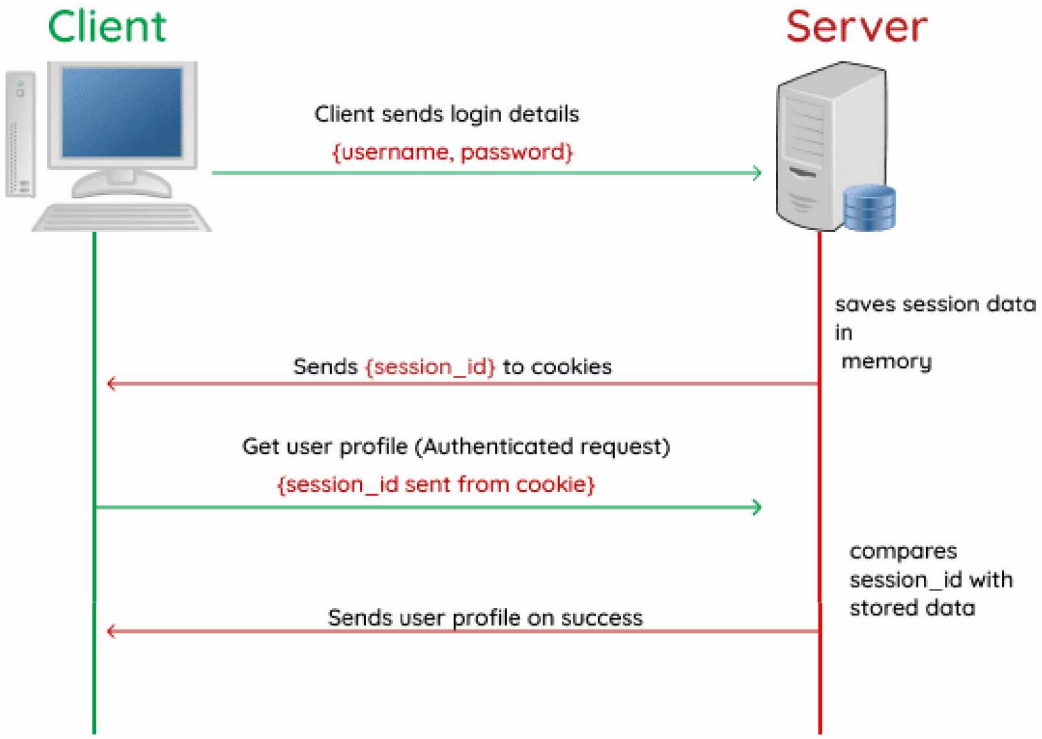
Mezi nejpoužívanější patří:

- session,
- JWT.

2.3.1 Session

HTTP je bezestavový protokol, nicméně takřka každá aplikace potřebuje stav mezi požadavky uchovávat [5]. A právě k tomu slouží session neboli relace.

Při autentizaci pomocí session každý uživatel, který vstoupí na stránku, obdrží jedinečný identifikátor *Session ID*, který se předává v cookies nebo v URL. Ten slouží jako klíč k session datům, kde lze uložit libovolné informace o přistupujících uživatelích. A to o každém zvlášť. Avšak doporučuje se neukládat příliš obsáhlé informace. Na rozdíl od cookies, které se uchovávají na straně prohlížeče, jsou data v session uchovávána na straně serveru [5].



Obrázek 8 – Diagram znázorňující autentizaci pomocí session. Zdroj: [36]

2.3.2 Session a zabezpečení

Hlavní zabezpečení session funguje na dvou hlavních specifikacích. Ta první říká, že identifikátor *Session ID* by měl mít podobu dostatečně dlouhého, náhodně generovaného klíče. To činí session odolnou vůči útoku hrubou silou. I dnes najdeme webové aplikace, které tento princip nedodržují. Často implementují generování session ve stylu čítače. V takovém případě stačí zkoušet například nižší hodnotu *Session ID,* než je ta naše. Pokud budeme úspěšní, dostaneme se do cizího účtu bez znalosti hesla a přihlašovacího jména.

Druhá specifikace doporučuje nastavovat session relativně krátkou dobu platnosti.

Taktéž se silně nedoporučuje přenášet *Session ID* v URL stránky, neboť působí proti principům SEO a ukládá se do historie webového prohlížeče.

Útok na uživatelovu *Session ID* ve webovém prohlížeči se jmenuje *Session hijacking*. O tomto útoku a jak se proti němu bránit pojednává kapitola 3.6.

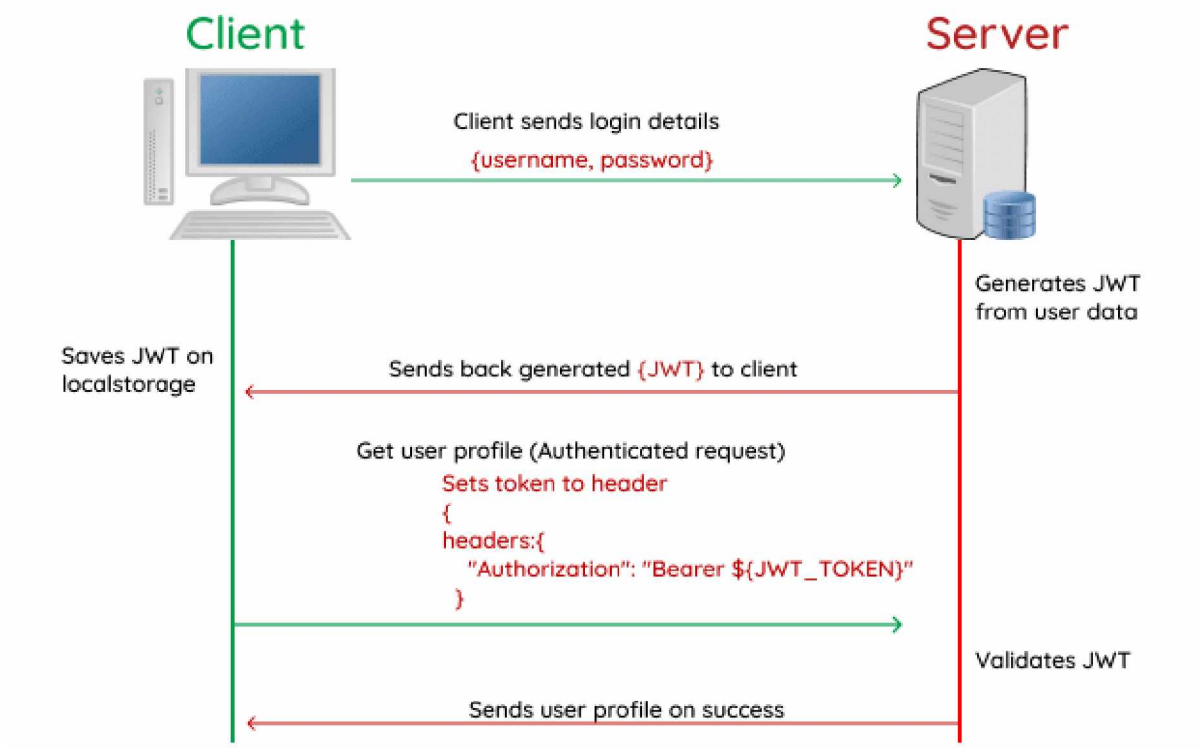
2.3.3 JWT

Session se dnes stále používá a pro spoustu aplikací je to naprosto dostačující mechanismus. Nicméně disponuje určitou nevýhodou. Jak již bylo řečeno, sessions se ukládají na straně serveru a využívají paměť. To se může stát problémem, když na server přistupuje příliš velký počet uživatelů.

Tento problém neplatí pro moderní autentizační systémy, které jsou založené na tzv. tokenech. A jednou z nejpoužívanějších implementací tokenů je JWT – *JSON Web Tokens*.

JSON Web Token je otevřený standard, který definuje kompaktní způsob bezpečného přenosu informací mezi stranami v podobě JSON objektu. Token se skládá ze tří částí oddělených tečkou [6]. Poté je uložen na straně klienta, což eliminuje nevýhodu session.

Tento způsob se někdy taktéž nazývá *Bearer Authentication*.



Obrázek 9 – Diagram znázorňující autentizaci pomocí tokenu JWT. Zdroj: [36]

Celý proces začíná prvotním přihlášením, kdy uživatel musí poskytnout přihlašovací údaje. Po úspěšném přihlášení vygeneruje server nový JWT token, který vrátí klientovi. Následně bude každý další požadavek na server obsahovat token v hlavičce *Authorization*, který je pokaždé validován. Tím je uživateli zaručen přístup ke chráněným zdrojům.

2.3.4 JWT a zabezpečení

Každá část tokenu je po vyplnění daty zakódována pomocí *Base64* [6]. Toto kódování převádí binární data do posloupnosti tisknutelných znaků, což skvěle ladí s přenosem dat skrze protokol HTTP.

Další stupně zabezpečení záleží na scénáři použití. Pokud potřebujeme zajistit integritu přenášených dat, lze token digitálně podepsat. Pro zajištění utajení mezi stranami lze token šifrovat. V případě autentizace se doporučuje využít obou možností.

Nicméně je potřeba aktivně sledovat vývoj kryptografických trendů, neboť náš používaný algoritmus pro šifrování či podepisování může být jednoho dne označen za prolomitelný. Tím nám vzniká bezpečností riziko, které je potřeba napravit volbou lepšího algoritmu.

2.4 Externí autentizační mechanismy

Do této skupiny patří takové mechanismy, které nejsou součástí samotné aplikace a využívají služby třetích stran. Patří sem například:

- OAuth 2.0,
- OpenID,
- Shibboleth,
- Kerberos,
- LDAP.

2.4.1 OAuth 2.0

Jedná se o otevřený protokol, který uživateli umožňuje udělit omezený přístup ke svým prostředkům na jednom či jiném webu, aniž by musel odhalit své přihlašovací údaje [7]. K získání přístupu ke chráněnému prostředku se používá přístupový token (tzv. *Access Token*). To je řetězec představující udělená oprávnění. Výchozí a nejčastěji používaný formát, pro tvorbu přístupového tokenu, je *JSON Web Token* (kapitola 2.3.3) [7]. Častým využitím je například funkcionalita přihlášení pomocí služby Facebook.

2.4.1.1 Role

V rámci protokolu definujeme následující role [7]:

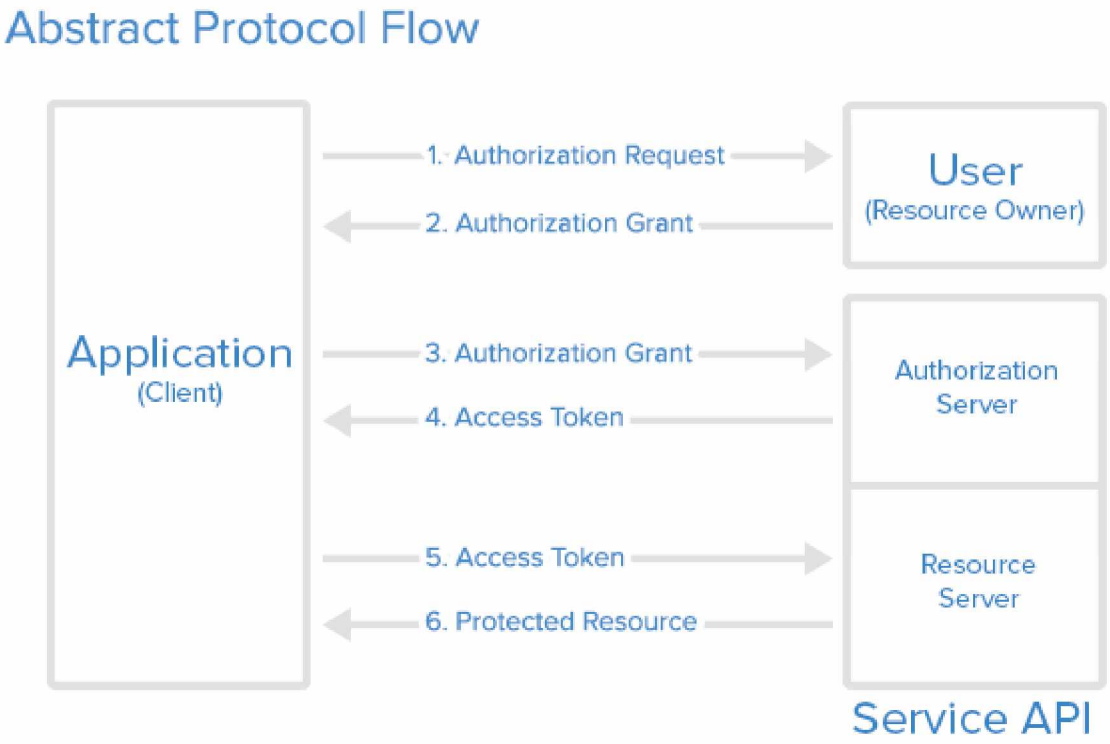
- Resource Owner – entita, která žádá o přístup ke chráněnému prostředku. Nejčastěji se jedná o koncového uživatele.

- Resource Server – server hostující chráněné prostředky. Například API, ke kterému chce uživatel přistupovat.
- Client – aplikace, která požaduje přístup k chráněnému prostředku ve jménu vlastníka zdroje.
- Authorization Server – server, který ověří vlastníka prostředků a vydá přístupové tokeny.

2.4.1.2 Tok protokolu

Obecný pohled na tok protokolu je následující [7]:

1. Aplikace požádá o autorizaci uživatele za účelem přístupu k prostředkům.
2. Uživatel autorizuje jeho přístup a aplikace obdrží pověření.
3. Aplikace požádá autorizační server o přístupový token.
4. Pokud je aplikace úspěšně autentizována a přístupové údaje uživatele jsou validní, autorizační server poskytne aplikaci přístupový token.
5. Aplikace požaduje přístup k chráněnému prostředku. Autentizuje se pomocí přístupového tokenu.
6. Pokud je přístupový token validní, server úspěšné odpoví na požadavek aplikace a poskytne chráněné prostředky.



Obrázek 10 – Diagram znázorňující obecný tok protokolu OAuth 2.0. Zdroj: [37]

2.5 Více faktorová autentizace

Doby, kdy stačilo ověření například pouze uživatelským jménem a hesel, jsou dávno pryč. Dnes je naprosto běžná více faktorová autentizace. Je to metoda ochrany přístupu k prostředku, která je založená na kombinaci zabezpečení ve více oblastech – faktorech.

Primárně se používají tyto faktory [8]:

- Znalost – něco, co uživatel zná.
- Vlastnictví – něco, co uživatel vlastní.
- Biometrie – něco, čím uživatel sám je.

Občas se taktéž mluví o faktorech *geolokace* a *čas*. Nicméně jejich primární použití je minimální, neboť je lze ve většině případů prolomit či obejít [8]. U specializovaných autentizačních systémů slouží spíše jako podpůrný faktor při rozhodování.

2.5.1 Faktor Znalost

Do tohoto faktoru patří prakticky cokoliv, na co si je uživatel schopen vzpomenout [8]. Například:

- heslo či PIN,
- osobní otázky,
- nakreslení gesta či znaku,
- jednorázové kódy.

2.5.2 Faktor Vlastnictví

Tento faktor většinou zahrnuje to, co je možné nosit v kapse či peněžence [8]. Lze zahrnout:

- Hardwarové tokeny – např. Yubikey.
- Platební karty.
- SIM karty.
- Osobní doklady.

2.5.3 Faktor Biometrie

Zde jsou na místě ty položky, které identifikují svého nositele:

- Čtečka otisku prstu.
- Sken oční sítnice.
- Rozpoznávání obličeje.
- CAPTCHA – speciální typ biometrie [8].

2.6 Hesla

Heslo je řetězec znaků, který se používá k ověřování identity při procesu autentizace. Pokud uživatel prokáže znalost hesla, je považován za oprávněného. Hesla nikdy neukládáme do databáze v čitelné podobě. Pokud by došlo k odcizení databáze, útočník má plný přístup ke všem účtům. Do databáze ukládáme pouze otisky hesel vytvořené pomocí dostatečně odolné hashovací funkce (více v kapitole 1.2). Poté porovnáváme pouze otisky.

Mezi nejčastější útoky na hesla patří:

- Útok hrubou silou – vyzkoušení všech možných kombinací.
- Slovníkový útok – vyzkoušení pravděpodobných hesel z předpřipraveného seznamu.
- Sociální inženýrství.
- Odchyt stisknutých kláves.
- Zachycení během přenosu.

Požadavky na odolné heslo se díky technologickým pokrokům neustále zvyšují. Nutno podotknout, že tyto požadavky se mění a na ty to změny je potřeba reagovat. Odolné heslo by mělo obsahovat [9]:

- Alespoň 8 znaků.
- Kombinace velkých a malých písmen.
- Číslice.
- Speciální znaky.

Pro každý účet bychom měli mít nové unikátní heslo. Praxe používání stejného hesla ve více, nebo všech účtech stále nevymizela. Tomu je potřeba se vyvarovat, neboť pokud se útočníkovi podaří heslo zjistit, získá přístup do všech účtů se stejným heslem. V případě používání unikátních hesel jsou ostatní účty stále chráněny.

Rozhodně bychom se měli vyvarovat krátkým jednoduchým heslům, která lze uhádnout:

- Primitivní heslo (např. heslo, 12345, admin).
- Jméno blízké osoby či domácího mazlíčka.
- Datum a osobní údaje.

Taktéž je dobré se vyvarovat postupům, který se jen tváří, že zvyšují bezpečnost hesla. Nicméně opak je pravdou. Patří sem:

- Číslo na konci hesla.
- První písmeno hesla velké.
- Triviální náhrady písmen (např. O za 0, A za 4).

Informovaný útočník dokáže tyto hesla jednoduše uhádnout pomocí slovníkového útoku. Dnes existují celé databáze výše zmíněných hesel, které útočník zkusí použít jako první. Šance na úspěch toho útoku je velká, neboť se vždy najde uživatel, který politiky pro tvorbu odolného hesla nedodržuje.

Jako další podpůrný faktor zabezpečení lze zavést černou listinu hesel. Ta obsahuje nejčastější používaná slabá hesla a nedovolí je uživateli použít. Tak se výrazně sníží šance na úspěšný slovníkový útok.

2.6.1 Správce hesel

Pokud se rozhodneme používat odolná hesla, nabízí se otázka: Jak si vše pamatovat? Lidský mozek není dobrý nástroj na pamatování a vymýšlení těchto hesel. A lepení lístečků s heslem na monitor už taky vyšlo z módy. Řešením je používání správce hesel.

Jedná se o speciální program, který si hesla pamatuje a vytváří za nás. Poté je potřeba si pamatovat pouze jedno silné hlavní heslo (tzv. *master password*), kterým se přihlašuje do programu. V současné době na trhu dominují tito poskytovatelé [10]:

- 1Password,
- LastPass,
- Dashlane,
- KeePass.

Většina těchto programů je placená a fungují jako cloudový systém. To poskytuje výhodu synchronizace mezi všemi zařízeními. Výjimkou je KeePass, který je sice zdarma, ale neposkytuje synchronizaci a podporu webové aplikace.

2.6.2 Funkcionalita Zapomenuté heslo

Jedná se o častou a velice oblíbenou funkcionalitu webových aplikací. Uživatel zapomene heslo od účtu a chce provést jeho změnu. Aplikace požádá o e-mail, který je spojený s uživatelským účtem a na ten zašle odkazy vedoucí k dalším krokům procesu. Pokud uživatel zadaným e-mailem skutečně disponuje, má k tomuto odkazu přístup.

Nicméně z vývojářského a bezpečnostního hlediska je nutné zajistit, aby tato funkcionalita byla co nejvíce uživatelsky přívětivá, ale zároveň bezpečná. Velmi jednoduše se totiž může přeměnit ve zbraň útočníka.

Začátek celého procesu se skládá ze situace, kdy se uživatel rozhodne o změnu hesla zažádat. Aplikace nabídne formulář, kam uživatel zadá svůj e-mail spojený s účtem. Aplikace tento e-mail ověří. Pokud v databázi existuje, odešle na e-mail informace s dalšími kroky a informuje uživatele o úspěšném provedení. V opačném případě taktéž informujeme uživatele o úspěšném provedení. Tím snižujeme množství informací, které potencionálnímu útočníkovi sdělujeme. Pokud bychom informovali, že zadaný e-mail v databázi neexistuje a nebyl odeslán, otevíráme tím možnost útočníkovi zjistit e-maily či účty, na které má smysl útočit. Informace o neúspěšném provedení vracíme pouze v případě výpadku - např. e-mailové služby.

V e-mailu vždy zasíláme:

- Odkaz pro změnu hesla.
- Odkaz pro zrušení procesu změny.

Odkaz pro změnu hesla nás přesměruje na stránku, kde si uživatel zvolí nové heslo a poté ho ještě jednou potvrdí, aby došlo ke shodě. Jelikož formulář odesílá citlivá data (heslo), je nutné zajistit, aby aplikace komunikovala pomocí zabezpečeného protokolu HTTPS (více v kapitole 4.1). Jinak může dojít k odposlechu dat. Pokud se aplikaci podaří změnu hesla dokončit, informuje uživatele a přesměruje ho např. na přihlašovací formulář.

Odkaz pro zrušení procesu změny je důležitý, když jsme o změnu hesla nezažádali, ale i přes to nám tento e-mail přišel. Například útočník zkouší náš e-mail. K tomuto případu slouží právě tento odkaz, který celý proces či žádost zruší. V případě častého opakování této situace je nejlepší kontaktovat administrátory aplikace a situaci s nimi řešit.

Do e-mailu nikdy neposíláme heslo v čitelné podobě.

Každá žádost o změnu hesla je identifikována náhodně vygenerovaným tokenem, který musí být dostatečně dlouhý, aby odolal případnému útoku hrubou silou. Pro náhodnost se v jazyce PHP doporučuje funkce *random_bytes()*, která generuje kryptograficky náhodné bajty. Poté už je jen potřeba pomocí funkce *bin2hex()* převést binární data na *ASCII* řetězec, který obsahuje hexadecimální reprezentaci vstupních dat.

```
$bytes = random_bytes(64); // generating 64 crypto random bytes
$token = bin2hex($bytes); // getting 128 (64 * 2) long string
```

Zdroj. kód 1 – Generování kryptograficky bezpečného tokenu v PHP

Zabezpečení lze zvýšit ukládáním pouze otisku tokenu do databáze. V takovém případě je potřeba zavést tzv. *selektor*. Jedná se taktéž o náhodně vygenerovaný řetězec, který ukládáme v čitelné podobě a stane se součástí celého tokenu odeslaného uživateli. Tento *selektor* slouží pouze pro vyhledávání v databázi. Zajišťuje bezpečnost funkcionality, neboť použití primárního klíče, který je velmi často sekvenčně generovaný, místo *selektoru*, velmi snižuje kryptografickou odolnost.

Jako poslední je potřeba uložit do databáze platnost tokenu. V praxi se doporučuje nastavit platnost ne vyšší, než 30 minut. Tento čas je nutné ukládat a porovnávat v časovém pásmu UTC. Díky tomu se vyhneme vážným bezpečnostním a implementačním problémům. Například pokud naše aplikace funguje ve více časových pásmech.

Celá validace se tedy skládá z následujících kroků:

1. Ověřit, zda záznam pod daným selektorem v databázi existuje.
2. Ověřit, zda už nevypršela platnost tokenu.
3. Ověřit shodu otisků tokenů.

Zabezpečení lze navýšit například použitím faktoru *znalost* (kapitola 2.5.1). Pokud naše aplikace nastavuje osobní otázky, lze se na ně v tomto procesu zeptat. Tím zvyšujeme důvěryhodnost uživatele a stěžujeme postup útočníka.

3 BEZPEČNOST WEBOVÉ APLIKACE

Dnešní svět bychom si bez webových aplikací dokázali jen stěží představit. Jejich využití šetří každý den čas a peníze. Bohužel ale ani webové aplikace nebyly ušetřeny před kybernetickými útoky a bezpečnostními dírami, na které si vývojáři těchto aplikací musí dávat pozor.

OWASP je nevýdělečná organizace založena v roce 2001 [7], která se snaží zlepšovat bezpečnost softwaru a webových aplikací. Nabízí popisy útoků a projekty pro edukaci a trénování moderní kyberbezpečnosti. Vydává publikaci OWASP Top 10, kde popisuje nejčastější typy chyb a útoků za poslední čtyři roky. Nejaktuálnějším vydání je z roku 2017 a nové vyjde v roce 2021.

I přes to, že dnešní doba nabízí využívání frameworků, kde většina bezpečnostních rizik je automaticky vyřešena, stále nevyřeší všechny. Tudíž by každý programátor měl mít dobré znalosti ohledně útocích a psaní bezpečného kódu.

3.1 Injection

Útok typu Injection je na prvním místě žebříčku OWASP Top 10 pro rok 2017 [8]. Základem tohoto útoku je neošetřený vstup od uživatele. Útočník se snaží odeslat data do aplikace takovým způsobem, který změní význam příkazů odesílaných interpretovi a změní tak očekávané chování aplikace. A jelikož tito interpreti často disponují vysokým oprávněním, úspěšný útok může vést k únikům citlivých dat, ztrátě kontroly nad webovým prohlížečem, aplikací a serverem. To činí z tohoto útoku velké bezpečnostní riziko.

Tato zranitelnost se vyskytuje ve velkém množství oblastí a technologií. Například:

- SQL/NoSQL dotazy,
- příkazy operačního systému,
- LDAP dotazy,
- XML a HTML dokumenty,
- HTTP hlavičky,
- JSON,
- URL,
- cesty k souborům.

3.1.1 Command Injection

Cílem tohoto útoku je vykonání libovolných příkazů na operačním systému hostitele skrze zranitelnou aplikaci. Tento útok je možný, když aplikace předává data uživatele, například skrze formulář nebo HTTP hlavičku, systémovému shellu. Shell poté tento příkaz vykoná se systémovým oprávněním aplikace. Zranitelnost vzniká díky nedostatečnému ošetření vstupu uživatele. Tudíž je potřeba vstup uživatele vždy korektně validovat.

Jako příklad útoku Command Injection nám poslouží aplikace, která provede příkaz *ping* na zadanou IP adresu.

```
// Missing input validation!
$target = $_POST['ip_address'];

/**
 * We have to determine OS.
 * On Windows(XAMPP) command Ping ends after four attempts.
 * On Unix we have to use switch -c to define number of attempts.
 */
if (stristr(php_uname('s'), 'Windows NT'))
    // Windows
    $result = shell_exec('ping ' . $target);
else
    // Unix - 4 attempts
    $result = shell_exec('ping -c 4 ' . $target);
```

Zdroj. kód 2 – Provedení příkazu operačního systému bez validace vstupu

Zdrojový kód č. 1 používá proměnou *target* od uživatele, aniž by provedl validaci, že se skutečně jedná o IP adresu. To umožňuje útočníkovi vložit vlastní škodlivý kód.

Na operačním systému Windows můžeme zkusit vložit do formuláře následující příkaz:

127.0.0.1 & ipconfig /all

Aplikace vypíše nejen výsledek příkazu *ping*, ale taktéž výstup příkazu *ipconfig*, který slouží k vypsání nastavení všech síťových rozhraní počítače. Tyto citlivé informace mohou být zneužity pro další útoky a napadení serveru samotného.

Na operačním systému typu Unix je potřeba vložit jiný příkaz:

127.0.0.1; ifconfig

```php
// Get input
$target = $_POST['ip_address'];
$target = stripslashes($target);

// Explode input to IP octets
$octet = explode('.', $target);

/**
 * If explode() returns array with length of 4
 * and all parts are numeric, we know there is an IP address in input.
 * Otherwise, it is not an IP address.
 */
if (count($octet) == 4 && is_numeric($octet[0]) && is_numeric($octet[1]) &&
is_numeric($octet[2]) && is_numeric($octet[3]))
{
    // Remake IP address from octets, who are numeric for 100%.
    $target = $octet[0] . '.' . $octet[1] . '.' . $octet[2] . '.' . $octet[3];

    /**
     * We have to determine OS.
     * On Windows(XAMPP) command Ping ends after four attempts.
     * On Unix we have to use switch -c to define number of attempts.
     */
    if (stristr(php_uname('s'), 'Windows NT'))
        // Windows
        $result = shell_exec('ping ' . $target);
    else
        // Unix - 4 attempts
        $result = shell_exec('ping -c 4 ' . $target);
}
else
    $result = 'Error: Invalid IP.';
```

Zdroj. kód 3 – Validace vstupů a bezpečné provedení příkazu

Zde už je vidět validace vstupů. Nejdříve si pomocí PHP funkce *explode* převedeme uživatelský vstup na pole, neboť víme, že validní IPv4 adresa se skládá ze čtyř oktetů rozdělených tečkou. Pokud vytvořené pole skutečně obsahuje pouze čtyři oktety a všechny jsou numerické, můžeme zpětně sestavit z oktetů IPv4 adresu a bezpečně provést příkaz *ping* se vstupem uživatele.

Pro moderní IPv6 adresu je tento příklad nefunkční. Ta se totiž zapisuje jako osm skupin čtyř hexadecimálních číslic a je možné ji v určitých případech zkracovat.

3.1.2 SQL Injection

Zde je cílem vložení vlastního SQL dotazu skrze vstup do aplikace. Pokud je útok úspěšný, následky jsou často fatální. Například ztráta či odcizení citlivých dat, narušení integrity dat, obejití autentizačních a autorizačních systémů a následné připojení do aplikace bez znalosti hesla. Zranitelnost znovu vzniká díky nedostatečnému ošetření vstupu uživatele.

Jako demonstrační příklad poslouží jednoduchá aplikace, která vyhledává z databáze zákazníků dle příjmení.

```
/**
 * Using prepared statements doesnt mean you cannot be
 * attacked by SQL Injection at all.
 * Here is a misuse of prepared statements.
 */
$statement = $connection->prepare(
        "SELECT * FROM customers WHERE lastname = '" . $_POST['lastname'] . "';");

$statement->execute();
```

Zdroj. kód 4 – Práce s databází umožňující SQL Injection

Zdrojový kód č. 3 znovu ukazuje, že se nevaliduje či neošetřuje uživatelský vstup. Ten je bez rozmyšlení použit jako součást SQL dotazu. Toto zanedbání způsobí bezpečnostní díru ve formě SQL Injection. Tu lze otestovat pomocí následujícího vstupu:

Doe' OR 1=1;--

Aplikace najednou změní své očekávané chování. Klauzule *WHERE* nyní závisí na dvou podmínkách. Jednak hledáme zákazníky, kteří mají příjmení Doe, a zároveň hledáme i ty zákazníky, kde platí, že 1=1. A jelikož 1=1 je vždy *TRUE*, aplikace nám nyní vrátí všechny cenné údaje o zákaznících. Došlo tedy k odcizení citlivých dat.

3.1.2.1 Obrana na straně aplikace

Obrana na straně aplikace je přímočará a efektivní. Jedná se o tzv. escapování znaků, které mají speciální význam v SQL. DBMS přesně specifikuje, které to jsou, což umožňuje vytvoření seznamu znaků, které je potřeba ošetřit.

K tomu se používají tzv. Prepared Statements. Jedná se o techniku pro vytváření SQL dotazů, která je odolná vůči SQL Injection.

```
/**
 * Correct using prepared statements with binding parameters
 * Successful protection against SQL Injection
 */
$statement = $connection->prepare(
    "SELECT * FROM customers WHERE lastname = :lastname");

$statement->bindParam(':lastname', $_POST['lastname']);
$statement->execute();
```

Zdroj. kód 5 – Korektní použití Prepared Statements

Uživatelský vstup nevkládáme do SQL dotazu přímo, ale pomocí tzv. „binding parameters". Poté zavoláním funkce *bindParam* řekneme, jaký parametr chceme nahradit skutečnou hodnotou. Tato hodnota je automaticky ošetřena proti SQL Injection [2].

I přes to, že tato obrana je relativně jednoduchá, spousta vývojářů na korektní ošetření stále často zapomíná. Taktéž je nutné podotknout, že používání Prepared Statements nás automaticky nechrání proti SQL Injection. Zdrojový kód č. 3 sice Prepared Statements používá, avšak nesprávným způsobem.

3.1.2.2 Obrana na straně databáze

Obrana na straně databáze spočívá zavedením vhodné hierarchie uživatelů a nastavením jejich oprávnění. Aplikace by tedy neměla používat výchozí administrátorský účet (tzv. „root") pro přístup do databáze, ale nově vytvořeného uživatele s takovým oprávněním, které skutečně potřebuje. Málokdy aplikace používá příkazy na úpravu tabulek (DROP, ALTER atp.). Stejně tak většinou nepotřebuje přístup k systémovým tabulkám daného databázového serveru, které jsou pro útočníka velmi cenné.

Toto zabezpečení nedokáže útoku na 100 % zabránit. Ve většině případů se jedná jen o extrémní ztížení. Nemělo by se tedy spoléhat jen na tento typ obrany. Nejlepším řešením je zavedení jak obrany na straně aplikace, tak i na straně databáze.

3.2 XSS

Při tomto útoku nejde o nic jiného než podstrčit vlastní nebezpečný script, který se vykoná v prohlížeči oběti. Poté útočník může získat plnou kontrolu nad prohlížečem, daty či účtem aplikace, aniž by oběť něco tušila.

XSS patří mezi nejznámější útoky. A přes to dnes najdeme webové stránky, které nejsou proti těmto útokům zabezpečeny. Útok dělíme na 3 typy:

- reflected,
- stored,
- DOM-based.

3.2.1 Reflected

Hlavním znakem tohoto typu je, že výsledek útoku vidíme ihned. Aplikace obdrží data a ty vloží do odpovědi bez ošetření. Nejčastěji se jedná o napadení části URL, která se poté interpretuje jako součást stránky.

Pro tento typ demonstrační aplikace funguje jako formulář, který slouží pro ohlašování závad aplikace. Ta zpětně uživateli jeho hlášení pro kontrolu vypíše.

```
// Validation of GET request
if (array_key_exists('problem', $_GET) && $_GET['problem'] != null)
    // Get input without protection
    $result = $_GET['problem'];
```

Zdroj. kód 6 – Neošetření vstupu proti Reflected XSS

Zdrojový kód č. 5 znovu disponuje proměnou *result*, která není ošetřena. A jelikož se jedná o *GET* požadavek, lze infikovat část URL vlastním zákeřným scriptem a zaslat ji oběti.

3.2.2 Stored

Jak už sám název napovídá, script se stane trvalou součástí obsahu stránky [2]. Nejčastěji vzniká, když je vstup ukládán do databáze a poté zpětně vypisován v HTML. Výhodou je, že pro přístup na takto napadené stránky není potřeba upravovat URL a donutit oběť, aby na odkaz klikla. Všichni uživatelé, kteří na napadenou stránku samovolně vstoupí, se stanou obětí tohoto útoku.

Tento typ útoku je taktéž nazýván jako *persistent* nebo *second-order* [8].

Pro ukázku útoku Stored XSS nám slouží aplikace návštěvní kniha, kde jakýkoliv uživatel může nechat hodnocení. Tato hodnocení jsou ukládána do databáze a poté vypisována.

```
$html = '<pre>';
foreach ($result as $row)
{
    // Missing escaping to protect against XSS attack
    $html .= $row["name"] . ': ' . $row["message"] . '<br>';
}
$html .= '</pre>';

echo $html;
```

Zdroj. kód 7 – Chybějící escapování dat při výpisu dat z databáze – Stored XSS

Při výpisu z návštěvní knihy se uložený uživatelský vstup neošetřuje, a tudíž vzniká bezpečnostní díra.

3.2.3 DOM-based

Typ DOM-based vzniká, když aplikace disponuje scriptem na straně uživatele, který zpracovává uživatelská data bez ošetření a poté je zapisuje zpět do DOM. Nedochází tedy ke generování škodlivého scriptu do stránky na straně serveru, a proto všechny obranné mechanismy na této straně selžou.

3.2.4 Obrana na straně serveru

Jediným správným řešením je ošetření pomocí PHP funkce *htmlspecialchars* při výpisu proměnných do HTML [2]. Jedná se o klíčovou funkci, která převede speciální znaky v textu na HTML entity. Mezi speciální znaky patři: &, ", ', <, >. Funkce by měla být použita při výpisu většiny proměnných [10].

```
// Validation of GET request
if (array_key_exists('problem', $_GET) && $_GET['problem'] != null)
    /**
     * Get treated input - this is the correct way in PHP
     * of protection against XSS attack
     * Third parameter encode depends on situation
     */
    $result = htmlspecialchars($_GET['problem'], ENT_QUOTES);
```

Zdroj. kód 8 – Korektní ochrana proti Reflected XSS pomocí funkce htmlspecialchars

```php
$html = '<pre>';
foreach ($result as $row)
{
    $html .= htmlspecialchars($row["name"], ENT_QUOTES) . ': ';
    $html .= htmlspecialchars($row["message"], ENT_QUOTES) . '<br>';
}
$html .= '</pre>';

echo $html;
```

Zdroj. kód 9 – Korektní ochrana proti Stored XSS pomocí funkce htmlspecialchars

Nyní lze namítat, že i když je ošetření jako takové jednoduché, tak jednou zapomenout v celé aplikaci použít funkci *htmlspecialchars* je taktéž jednoduché. Vývojář je člověk, který taky zapomíná.

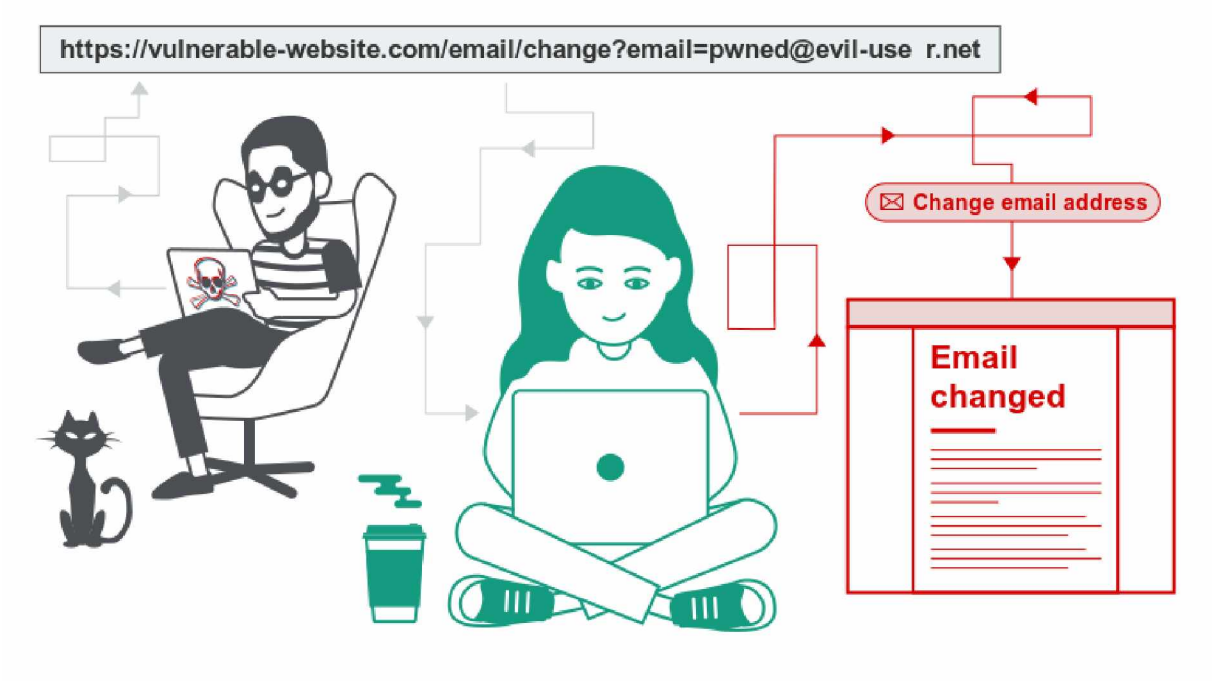
Tato skutečnost vedla k zavedení další úrovně zabezpečení proti XSS. Řeč je o šablonovacích systémech pro PHP, které jsou dnes součástí všech moderních frameworků. Tyto systémy většinou disponují vlastní syntaxí a automaticky řeší bezpečnost za nás tím nejlepším způsobem. Avšak i zde lze při nesprávném použití způsobit bezpečnostní díru. Mezi nejznámější reprezentanty patří *Latte* [11] a *Twig* [12].

3.2.5 Obrana na straně uživatele

Dříve bylo jednou z technik zakázat Javascript. Ale význam Javascriptu mezitím rapidně vzrostl a dnes je tato obrana prakticky nemožná, neboť je dnes běžnou a neodmyslitelnou součástí webových stránek. Avšak moderní technologie a webové prohlížeče nabízejí efektivní obranu na straně uživatele i se zapnutým Javascriptem. Podrobně v kapitole 4.2.

3.3 CSRF

Cross-site request forgery je zranitelnost webové aplikace, která útočníkovi umožňuje přimět uživatele k provedení akce, kterou nemá v úmyslu provést [17]. Tento útok je často veden proti aplikacím, které pro uchování informace přihlášení uživatele využívají cookie nebo hlavičku *Authorization* [18].



Obrázek 11 – Ukázka útoku CSRF. Zdroj: [39]

Transparentní navštívení aplikace se dá zařídit například pomocí HTML elementu img. Tomu dáme do parametru *src* URL stránky, kterou chceme, aby uživatel neúmyslně navštívil [18]. Poté musíme nalákat naši oběť na stránku, kam jsme vložili výše zmíněný HTML element. Prohlížeč potom spolu s požadavkem na danou URL pošle i autorizační cookie (např. *Session ID*). Server sice vrátí prosté HTML, ale na to prohlížeč zareaguje zobrazením ikony pro nefunkční obrázek [18]. Pokud obrázek skryjeme, oběť si ničeho nevšimne.

Útok CSRF je jednou z aplikací problému zmateného zástupce [19].

3.3.1 Problém zmateného zástupce

Problémem zmateného zástupce označujeme takovou situaci, kdy strůjcem škodlivé akce je sám uživatel, nikoliv útočník [19]. Uživatel o provedených změnách vůbec netuší. Nejčastěji se s touto situací setkáme právě v informatice.

V angličtině tento problém najdeme pod názvem *Confused deputy problem* [19].

3.3.2 Obrana

Je potřeba si uvědomit, že obranu má smysl zavádět v případech, kdy operace či akce provádí změnu stavu aplikace. Pokud bychom vedli CSRF útok na URL, která pouze zobrazuje HTML či vrací data, nemá tento útok smysl, neboť útočník neuvidí odpověď serveru [1].

A to díky zabezpečovacímu mechanismu *Same-origin policy* webových prohlížečů. Tento mechanismus umožňuje webové stránce přístup k datům na jiné webové stránce pouze v případě, že obě webové stránky mají stejný původ (tzv. *origin*) [20]. Dvě webové stránky mají stejný původ, když mají stejný protokol, port a host. Nutno podotknout, že tento mechanismus se aplikuje pouze na skripty. To zabraňuje škodlivému skriptu čtení citlivých dat na jiné webové stránce. Zdroje jako obrázky a kaskádové styly tomuto mechanismu nepodléhají [20].

Avšak pokud určitá URL mění stav aplikace, je potřeba ji zabezpečit. A to bez ohledu na typ HTTP metody. Nejčastějším případem je administrátorská část webové aplikace, kde se mažou záznamy a upravuje nastavení [17].

Pro zabezpečení slouží tzv. CSRF token. Jedná se o náhodně vygenerovaný řetězec, který platí pouze pro jednu akci a pro jednoho aktuálního uživatele, který by se měl po každém požadavku měnit [17]. Aplikace si token zapamatuje (např. do session) a zároveň ho uloží do vygenerovaného HTML dokumentu. Pokud dojde ke zpracování akce, je v požadavku odeslán i token z HTML, který je poté porovnán s uloženým tokenem na straně serveru. Při shodě můžeme akci zpracovat. V opačném případě víme, že se jedná o CSRF útok. Síla této obrany tkví v tom, že útočník nemá šanci CSRF token předem uhodnout, neboť je generován zcela náhodně [1].

3.4 Clickjacking

Tento způsob útoku cílí na uživatele webových stránek a rozhraní webových prohlížečů [21]. Nejprve útočník naláká uživatele na stránku s neškodným obsahem. Nicméně do této stránky je taktéž vložena jiná stránka (nejčastěji pomocí HTML elementu iframe). Ta je zobrazena na pozadí se zapnutou průhledností, takže o ní uživatel neví. Když poté uživatel kliká, ve skutečnosti kliká na neviditelnou stránku navrchu. Díky tomu může uživatel provést akci či požadavek, aniž by o tom věděl.

Tento útok se taktéž někdy nazývá *UI redressing* [21] a je taktéž aplikací problému zmateného zástupce.

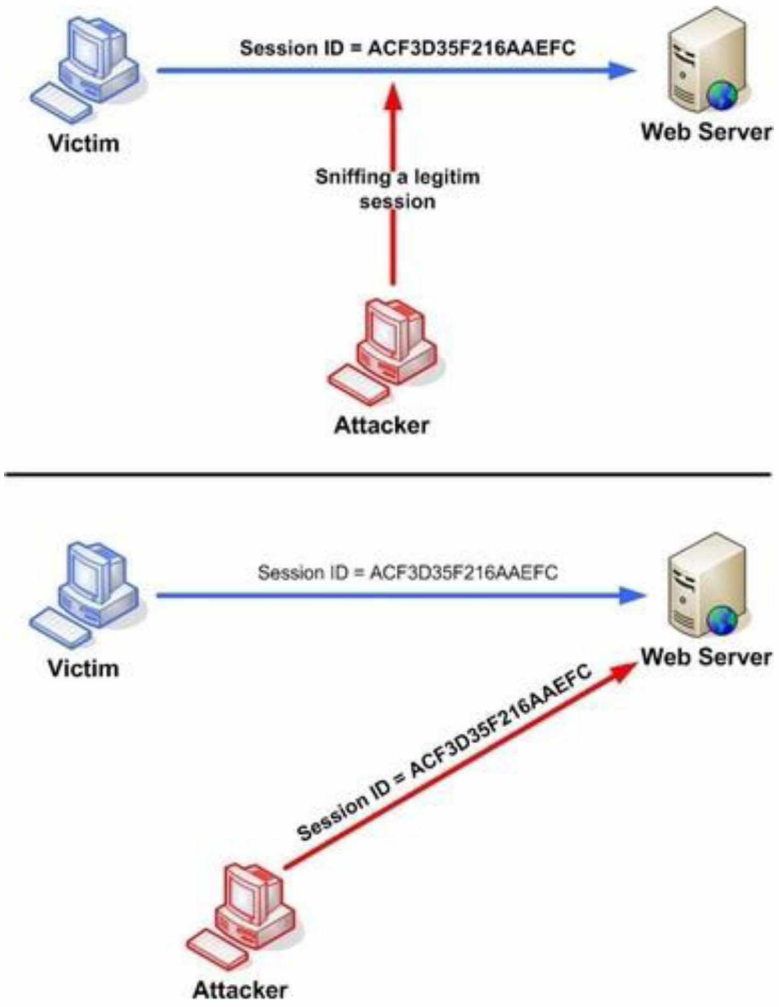
3.4.1 Obrana

Efektivní obrana je dvojí. První začíná u uživatele samotného a spočívá v používání doplňků prohlížečů. Mezi nejznámější a nejpoužívanější patří program NoScript [22]. Ten je zcela zdarma a poskytuje dodatečné zabezpečení v prohlížečích založených na jádře webového prohlížeče Mozilla [22]. Tento program automaticky blokuje jakýkoliv obsah ve formě Javascriptu, Javy a Adobe Flash. Poté ho spustí, jakmile to uživatel povolí.

Druhá spočívá v zavedení bezpečnostních hlaviček na straně serveru. O tomto tématu podrobně pojednává kapitola 4.2.

3.5 Session Hijacking

Jedná se o způsob útoku, který využívá vlastnosti protokolu HTTP k odcizení cookies oběti za účelem získání neoprávněného přístupu k informacím nebo službám webového serveru [23]. Nejčastěji se jedná o krádež cookie *Session ID*, která se používá u autentizace webových aplikací pomocí session mechanismu. Charakteristika session byla již probrána v kapitole 2.3.1.



Obrázek 12 – Diagram znázorňující útok typu Session Hijacking. Zdroj: [38]

Existuje 5 základních metod, jak tento útok provést [23]:

- session fixation,
- session side-jacking,
- cross-site scripting (XSS),
- malware,
- útok hrubou silou.

3.5.1 Session fixation

U této metody si útočník sám nastaví hodnotu *Session ID* a tu poté zašle oběti jako součást odkazu [23]. Poté čeká, dokud se uživatel nepřihlásí pomocí zaslaného odkazu. Pokud se uživatel přihlásí, zaslané *Session ID* se stane platným a útočník ho může začít používat.

```
<a href="https://www.VulnerableSite.com/login.php?sessionid=mySessionID">
  Log in!
</a>
```

Zdroj. kód 10 – Ukázka HTML pro útok metodou Session fixation

3.5.2 Session side-jacking

Celá metoda spočívá v odposlouchávání nešifrované komunikace mezi prohlížečem a webovým serverem. A to pomocí speciálních programů, které dokáží analyzovat síťový provoz [23]. *Session ID* je uložena v cookie, který se posílá jako součást HTTP požadavku. A jelikož při nešifrované komunikaci se v síti přenáší čistý text, útočník má hodnotu *Session ID* plně k dispozici.

3.5.3 Cross-site scripting (XSS)

Nejsilnější a nejnebezpečnější metoda spočívá ve využití útoku XSS. Pokud cílová aplikace disponuje zranitelností typu XSS, lze ji efektivně využít pro odcizení hodnoty *Session ID*. Útočník donutí uživatele otevřít odkaz, ve kterém je již vložený vlastní kód (nejčastěji Javascript). Po kliknutí na odkaz a načtení stránky se provede taktéž i útočníkův škodlivý kód. Ten může zcizit například všechny hodnoty cookie, které jsou s cílovou aplikací spojené. A to včetně *Session ID*.

3.5.4 Malware

Jednou z velmi efektivních metod je donutit uživatele nainstalovat na své zařízení tzv. malware. To je typ programu, který je určený k cílenému poškození zařízení oběti. Po instalaci začne provádět automatické procházení obsahu zařízení. A to včetně hledání našeho *Session ID*. Například tím, že bude aktivně sledovat naši komunikaci po síti a poté data odesílat útočníkovi.

3.5.5 Útok hrubou silou

Útočník může vždy zkusit uhodnout hodnotu *Session ID* uživatele. Nicméně tato metoda má smysl, pokud aplikace používá krátké a předvídatelné hodnoty.

3.5.6 Obrana

Obranu proti Session Hijacking lze popsat souhrnně. Dodržení a nastavení následujících pravidel ochrání aplikaci proti všem metodám útoku.

Nejdůležitějším krokem je zajištění šifrované komunikace pomocí protokolu HTTPS v celém rozsahu aplikace. To zabrání útočníkovi číst hodnotu našeho *Session ID* v čitelné podobě.

Nutno podotknout, že protokol HTTPS je sám o sobě velké téma a podrobně je rozebrán v kapitole 4.1.

Dále je potřeba nastavit správné atributy pro cookie, které se nastavují v HTTP hlavičce *Set-Cookie* [23]. To ochrání před útoky typu XSS a CSRF.

Tyto atributy jsou:

- HttpOnly – zabraňuje skriptům na straně uživatele k přístupu cookie.
- Secure – cookie je zaslána pouze přes šifrovaný protokol HTTPS.
- SameSite – cookie je zasílána pouze při komunikaci se stránkou, ze které pochází.

Nakonec je potřeba zajistit, aby se vygenerovala nová *Session ID* při přihlášení uživatele, a aby hodnota byla odolná vůči útoku hrubou silou. Moderní frameworky a knihovny disponují mechanismy, které na tyto slabiny myslí a automaticky je za nás řeší.

Proti útoku pomocí malwaru není lepší obrana než lidský rozum a aktualizovaný antivirový program, který dokáže tyto programy analyzovat a eliminovat.

3.6 Security.txt

Až do teď byla rozebírána obrana proti již uvedeným druhům útoků. Ale co dělat v případě, když například nezávislý bezpečnostní expert najde bezpečnostní riziko na webové stránce a chce to oznámit osobě, která má řešení těchto situací na svědomí? Nutno podotknout, že s touto informací se musí nakládat opatrně. Pokud by se dostala do rukou útočníka, mohl by bezpečnostní riziko využít dříve, než bude opraveno. Dosud neexistoval jednotný způsob, jak se s touto situací vypořádat.

Nyní se tuto situaci snaží vyřešit nový standard *security.txt* [24]. Jedná se o soubor, ve kterém se čtenář dozví kam a jak případné chyby či bezpečnostní rizika hlásit [24]. Soubor se musí nacházet v adresáři */.well-known/*. Tento adresář je definován v RFC 5785 [25] a slouží jako kořenový adresář pro všechny *„.well-known“* umístění. Aktuálně standard obsahuje osm direktiv, které slouží k popsání komunikačních kanálů a podobných informací. Nutno zajistit, aby tento soubor a odkazy u direktiv byly přístupné přes protokol HTTPS (kapitola 4.1).

Cesta k *security.txt* souboru by tedy měla být:

https://www.example.com/.well-known/security.txt

Standard taktéž definuje, že soubor by měl být přístupný i na této adrese:

https://www.example.com/security.txt

V tomto případě se musí jednat o přesměrování na odkaz s *„.well-known“* adresářem.

3.6.1 Contact

Direktiva *Contact* je nejdůležitější část, která je jako jediná povinná [24]. Pomocí ní sdělujeme komunikační kanál, který je vyhrazený pro oznamování bezpečnostních rizik. Těchto kanálů může být víc a měli by být seřazeny dle toho, jaké kanály upřednostňujeme. Lze uvést například e-mail, telefonní číslo, webovou stránku či účet na sociální síti. Jak již bylo řečeno, všechny odkazy na web musí obsahovat HTTPS. V případě e-mailu a telefonního čísla musíme přidat URI schéma *mailto:* a *tel:*.

```
Contact: https://www.example.com/security-contact
```

Zdroj. kód 11 – Direktiva *Contact*

3.6.2 Encryption

Direktiva označuje šifrovací klíč, který by se měl použít pro šifrovanou komunikaci v průběhu oznamování bezpečnostního rizika. Využívá se asymetrické šifrování. Direktiva neočekává přímo klíč, ale odkaz na místo, kde se klíč nachází. Direktiva je volitelná.

```
Encryption: https://www.example.com/pgp-key.txt
```

Zdroj. kód 12 – Direktiva *Encryption*. Zdroj: [24]

3.6.3 Acknowledgments

V této direktivě můžeme vložit odkaz na místo, kde jsou uvedeni lidé, kteří našli bezpečnostní riziko. Spolu s poděkováním a informací, jaké riziko odhalili. Nicméně bychom měli být opatrní, kolik informací sdělujeme. Čím méně toho sdělíme o odhaleném riziku, tím lépe. Jedná se o volitelnou direktivu.

```
Acknowledgments: https://www.example.com/hall-of-fame
```

Zdroj. kód 13 – Direktiva *Acknowledgments*

3.6.4 Canonical

Direktiva označuje kanonickou URI, kde se soubor nachází. Účelem této direktivy je možnost aplikace digitální podpisu na lokaci souboru. Taktéž se jedná o ověření, že nejde o zcizený soubor.

```
Canonical: https://www.example.com/.well-known/security.txt
```

Zdroj. kód 14 – Direktiva *Canonical*

3.6.5 Policy

Direktiva *Policy* je volitelná a obsahuje odkaz na místo, kde subjekt sděluje svoji politiku pro oznamování bezpečnostních rizik.

```
Policy: https://www.example.com/security-policy
```

Zdroj. kód 15 – Direktiva *Policy*

3.6.6 Hiring

Hiring direktiva slouží pro sdělování volných pracovních míst souvisejících s bezpečností. Direktiva je volitelná a může se v souboru objevit vícekrát (např. více kanálů pro sdělování pracovních míst).

```
Hiring: https://www.example.com/security-jobs
```

Zdroj. kód 16 – Direktiva *Hiring*

3.6.7 Expires

Tato direktiva udává datum a čas, po kterém již soubor není platný a neměl by být použit. Direktiva se smí v souboru použít pouze jednou a je volitelná.

```
Expires: Thu, 31 Dec 2020 18:37:07 -0800
```

Zdroj. kód 17 – Direktiva *Expires*. Zdroj: [24]

3.6.8 Preferred-Languages

Direktiva slouží pro definování jazyka, kterým jsme schopni komunikovat při oznamování bezpečnostního rizika. Direktiva může obsahovat více hodnot (jazyků), které jsou oddělené čárkami. Hodnoty jazyků jsou značky dle RFC 5646 [24]. Pokud je direktiva použita, musí mít alespoň jednu hodnotu. V opačném případě předpokládáme, že očekáváme komunikaci v anglickém jazyce. Uvedené jazyky nejsou seřazeny dle priority. Zodpovědná osoba musí být schopna efektivně komunikovat v jakémkoliv uvedeném jazyce.

Direktiva je volitelná a může být v souboru maximálně jednou.

```
Preferred-Languages: en, cz
```

Zdroj. kód 18 – Direktiva *Preferred-Languages*

4 BEZPEČNOST SÍŤOVÉHO STACKU

Jako poslední je potřeba zabezpečit komunikaci mezi uživatelem a serverem. Nezabezpečená komunikace je terčem útoku MITM – Man in the middle. Jedná se o pasivní útok, kdy útočník je schopen odposlouchávat celou komunikaci, a to včetně důležitých informací jako hesla, *Session ID* či tokenu pro změnu hesla [26]. Vzniká například připojením na Wi-Fi, která je v rukou útočníka. Webový prohlížeč nekomunikuje se serverem, ale s útočníkem. Ten poté celou komunikaci posílá dál a vrací odpovědi ze serveru. Díky tomu má útočník celou komunikaci k dispozici a uživatel nic nepozná.

Dnes je zabezpečení komunikace pomocí protokolu HTTPS již standardem. Pokud aplikace toto zabezpečení ignoruje, ohrožuje své uživatele a snižuje svoji důvěryhodnost.

4.1 HTTPS

Tento protokol, celým názvem Hypertext Transfer Protocol Secure, se v informatice používá pro šifrovanou komunikaci, nejčastěji mezi webovým prohlížečem a serverem. Jedná se o spojení protokolů HTTP a SSL/TLS [26].

SSL je protokol, který se používá pro šifrování a zabezpečení dat. Využívá asymetrickou kryptografii, která je založena na využívání certifikátů. Nicméně je již velmi starý a obsahuje zranitelnosti, díky kterým je útočník schopen šifrovanou komunikaci prolomit. Z těchto důvodů se již nedoporučuje používat. Jeho nástupcem je moderní protokol TLS, jehož nejnovější verze je 1.3.

HTTPS komunikaci lze jednoduše popsat tak, že klient a server nejdříve mezi sebou vytvoří šifrovaný tunel, a poté skrze něj komunikují pomocí standardního HTTP [26]. Při vytváření tunelu musí klient sdělit název domény (např. example.com), se kterou se chce spojit. To je jediná informace, která se odesílá nezašifrovaná. Zbytek URL se posílá až do šifrovaného tunelu [26]. Útočník je tedy schopen monitorovat jaké webové servery uživatel navštěvuje, ale neví, co tam přesně dělá. Standardní číslo portu pro HTTPS je 443.

4.1.1 Certificate authority (CA)

Certifikační autorita je společnost, která vydává a validuje elektronicky podepsané veřejné šifrovací klíče, kterým se říká digitální certifikáty. Zároveň tím prokazuje vlastnictví domény. Běžná platnost certifikátu je jeden rok a musí se zaplatit. Jeho vystavení a instalace do prostředí webového serveru je manuální záležitost.

Webové prohlížeče respektují přibližně 400 důvěryhodných certifikačních autorit. Mezi nejznámější patří [26]:

- Google CA,
- COMODO,
- RapidSSL.

Avšak největší popularitě se poslední dobou těší certifikační autorita Let's Encrypt. Ta poskytuje certifikáty zdarma, které mají platnost 90 dní. Celý proces nastavení a obnovení certifikátu lze automatizovat pomocí klienta *Certbot*. V případě blížící se expirace informuje vlastníka o vypršení certifikátu.

4.1.2 Certificate Transparency (CT)

Aby byla činnost certifikačních autorit co nejvíce pod kontrolou, musí být všechny platné certifikáty vystaveny v databázích Certificate Transparency. Jedná se o log vydaných certifikátů [26]. Aby prohlížeč považoval certifikát za platný, musí dostat potvrzení, že certifikát byl skutečně vložen do logů CT. V opačném případě vyhodnotí certifikát jako neplatný a stránku nezobrazí.

Jako důkaz slouží tzv. SCT – Signed Certificate Timestamp. To je digitálně podepsané časové razítko, kdy byl certifikát vložen do logů CT. Toto razítko nejčastěji zařizuje sama certifikační autorita. Ta při tvorbě certifikátu zároveň zajistí vložení do logů CT a přidá hodnotu SCT do certifikátu [26].

Díky CT je možné hlídat domény před zneužitím. Existují totiž nástroje, které tyto logy monitorují. V případě, že byl vydán nový certifikát pro danou doménu, vlastník domény je ihned informován. Pokud k obnově nemělo dojít, lze předpokládat, že se někdo snaží zneužít doménu např. pro phising. Vlastník může díky monitoringu ihned zareagovat a své klienty varovat.

V současné době dominují tyto služby [26]:

- Certificate Transparency Monitoring – Facebook for Developers,
- Certificate Search,
- Suricat.

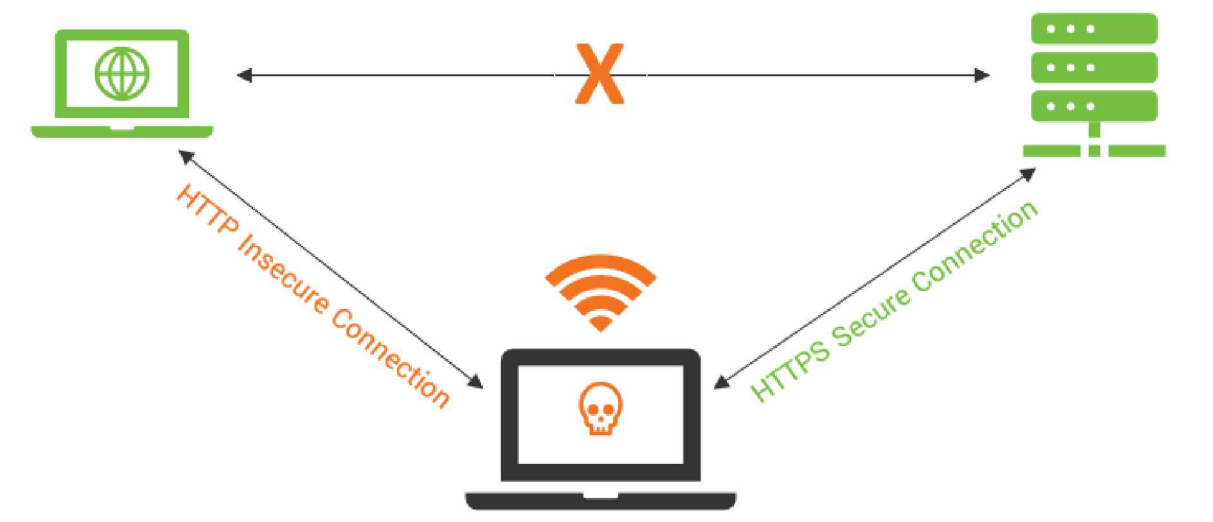
4.1.3 TOFU a SSL Stripping

TOFU neboli Trust-on-First-Use je v informatice bezpečnostní model, který je využíván k navázání důvěryhodné relace s dosud nedůvěryhodnou entitou [26]. Uživatel musí rozhodnout, zda lze této dosud neznámé entitě věřit. Jako příklad lze uvést připojování na server pomocí SFTP klienta. Pokud je na server přistupováno poprvé a jeho identifikátor (např. veřejný klíč) není uložen v lokální databázi, klient donutí uživatele rozhodnout, zda nové entitě (serveru) lze věřit a navázat s ní bezpečné spojení.

Na tomto modelu je postaven i protokol HTTPS. Současné chování moderních prohlížečů je takové, že výchozí protokol pro komunikaci je stále HTTP. Pokud je cílová doména nastavena pouze pro šifrovanou komunikaci (HTTPS), vrátí stavový kód 302 pro přesměrování na HTTPS. Až teprve poté webový prohlížeč vytváří šifrované spojení.

Část komunikace, kdy prohlížeč stále komunikuje pomocí nešifrovaného HTTP lze využít pro útok typu SSL Stripping. Je to aplikace útoku MITM, která se dá použít proti šifrované komunikaci [26]. Využívá skutečnosti, že většina uživatelů internetu nehlídá, zda komunikují přes HTTPS. Prohlížeč je sice varuje, že komunikace není šifrovaná, ale uživatel to stejně ignoruje.

Útočník opět vidí veškerou komunikaci. Rozdíl oproti MITM je, že útočník se serverem komunikuje korektně přes HTTPS. Odpovědi serveru odešle zpět uživateli ve formě HTTP.



Obrázek 13 – SSL Stripping. Zdroj: [40]

4.1.4 Útoky a zabezpečení

Za dobu existence HTTPS bylo oznámeno již několik útoků na tento protokol. Avšak rozbor těchto útoků není součástí této práce. Na obranu stačí dodržovat nejnovější doporučená nastavení protokolu.

Mezi útoky patří například [26]:

- BEAST,
- RC4,
- POODLE,
- Logjam,
- FREAK,
- ROBOT,
- Heartbleed.

Pro zabezpečené HTTPS se již nesmí používat protokol SSL, ale jeho náhrada TLS. I tento protokol se vyvíjí, přičemž doporučené verze jsou dnes 1.2 a 1.3 [26]. Ty bezpečně ochrání před výše zmíněnými útoky [26]. Pokud je nutné z dobrých důvodů zachovat kompatibilitu se staršími zařízeními či knihovnami, lze stále používat starší verze TLS.

Pro vygenerování aktuálního doporučeného nastavení lze použít SSL Configuration Generator od společnosti Mozilla [27]. Ten obsahuje různé druhy nastavení pro různý software, a to včetně kryptografických šifer. Pokud tedy člověk není zběhlý administrátor, který se v tomto nastavení perfektně vyzná, může si díky této službě ulehčit práci.

Pro následující kontrolu nastavení HTTPS je dobré využít službu SSL Labs společnosti Qualys [28]. Ta po zadání domény stránku kompletně analyzuje. Výstupem je známka a kompletní rozbor.

Jako další stupeň zabezpečení by se měla zavést hlavička HSTS a s ní spojený tzv. preload list. Ten zajistí automatické využívání HTTPS u klienta a dokáže ochránit proti útoku SSL Stripping. Více o této hlavičce v kapitole 4.2.2.

Na závěr pro osobní ochranu lze doporučit rozšíření webového prohlížeče HTTPS Everywhere. Toto rozšíření je k dispozici pro drtivou většinu moderních prohlížečů a nutí prohlížeč automaticky provádět šifrované požadavky přes HTTPS [29].

4.2 Security Headers

Security Headers neboli bezpečnostní hlavičky jsou podmnožinou HTTP hlaviček, které nastavují bezpečnostní pravidla mezi webovým prohlížečem a serverem [30]. Pomocí těchto hlaviček jsme schopni povolit nebo zakázat určité funkce prohlížeče za účelem vyšší bezpečnosti a soukromí. Jedná se o moderní způsob aktivní ochrany webových stránek a aplikací [30].

4.2.1 Content-Security-Policy

Content-Security-Policy (CSP) je nejdůležitější bezpečností hlavička, která nastavuje politiku načítání zdrojů do webových stránek [30]. Přesně definuje, jaké soubory lze do webových stránek nahrát a jakým způsobem. Toto nastavení lze rozčlenit až na jednotlivé typy souborů (např. obrázky, skripty, styly, fonty, média) [30]. Taktéž umožňuje načítání obsahu pouze přes HTTPS nebo z vlastní domény.

Z pohledu bezpečnosti webových aplikací je hlavička velice efektivní proti útokům XSS a Clickjacking. Pomocí ní lze přesně nadefinovat povolené URL, ze kterých může webový prohlížeč stahovat Javascript a ostatní zdroje. Taktéž lze zakázat inline scripty. To eliminuje možnost útoku XSS. V neposlední řadě lze taktéž zakázat rámování stránky a zabránit tím útoku typu Clickjacking.

Nicméně i přes to, že hlavička je velice efektivní, může být obsáhlá a její nasazení u velkých projektů náročné. Je potřeba analyzovat načítání veškerého obsahu, který aplikace používá. Poté politiku CSP budovat a testovat. Pokud by byla analýza chybná, nemusí se všechny zdroje korektně načíst, což by vedlo k poškozenému zobrazení webové stránky. V takovém případě prohlížeč informuje do vývojářské konzole [30].

Pro testovací a implementační účely se používá tato hlavička – CSP Report Only. Jedná se o reportovací verzi hlavičky CSP, jenž umožňuje testovat aktuální nastavení CSP. Prohlížeč hlavičku obdrží, zpracuje ji a bude se řídit definovanými pravidly. Avšak místo exekuce zajistí zpětnou vazbu o výsledku nastavení ve vývojářské konzoli [30].

4.2.2 HTTP Strict-Transport-Security

Tato hlavička prohlížeči přikazuje, že má po určitou dobu komunikovat se serverem pouze přes HTTPS protokol, lze využít tuto hlavičku (ve zkratce HSTS) [30]. Nutí uživatele automaticky používat šifrovanou komunikaci, což zvyšuje celkovou bezpečnosti aplikace. Hlavička je odesílána pouze přes šifrované spojení HTTPS [26].

S HSTS hlavičkou se pojí tzv. preload list. Jedná se o seznam domén, na které má webový prohlížeč přistupovat pomocí protokolu HTTPS automaticky už při prvním spojení. Díky tomuto seznamu lze efektivně předcházet útoku typu SSL Stripping a problematice TOFU. To samozřejmě platí pro ty prohlížeče, které HSTS plně podporují.

Do seznamu se registruje doména přidáním direktivy preload do HSTS hlavičky, a poté se žádá o přidání na stránce https://hstspreload.org/. Tato stránka je spravována společností Google. Při žádosti o přidání dojde k validaci, že se direktiva preload skutečně nachází v hlavičce HSTS. Tento mechanismus kontroluje, že o přidání skutečně žádá vlastník domény, a nejedná se o útočníka nebo konkurenci, jejíž záměr je uškodit. Po úspěšné kontrole bude doména přidána do seznamu. Tento seznam, který je interní součástí prohlížeče, se obnovuje až s novou verzí prohlížeče. Disponují jím všechny moderní prohlížeče, které používají výše zmíněný zdrojový seznam od společnosti Google [31].

Využívání preload listu a HSTS hlavičky se samozřejmě z pohledu bezpečnosti velice doporučuje, ale je potřeba se ujistit, že server je opravdu schopen a připraven nepřetržitě používat HTTPS. Jakmile by SSL certifikát přestal platit či by došlo k jiné chybě, uživatelé nebudou schopni se dostat na webovou stránku, neboť webový prohlížeč bude stále vyžadovat HTTPS komunikaci, kterou server není schopen poskytnout. Tím vzniká škoda. Zažádat o odejmutí z preload listu je možné, ale seznamy v prohlížečích budou opět obnoveny až s novou verzí prohlížeče [31].

Hlavička disponuje třemi direktivami [30]:

- max-age – nastavení doby (v sekundách), pro komunikaci pouze přes HTTPS.
- includeSubDomains – nastaví, že i subdomény musí disponovat SSL certifikátem.
- preload – potvrzení, že skutečně chceme naši doménu přidat do preload listu.

```
# To access web page, use HTTPS for one year
# For all subdomains
# Confirming preload
Strict-Transport-Security: max-age=31536000; includeSubDomains; preload
```

Zdroj. kód 19 – Nastavení hlavičky HSTS

4.2.3 Referrer-Policy

Tato hlavička má na starosti zabezpečení běžné HTTP hlavičky *Referer*, obsahující adresu webové stránky, ze které uživatel přišel [32]. Tato informace může být použita například pro analytiku provozu, nebo logování. I když tato hlavička vypadá nevinně, nese určité bezpečnostní riziko.

Ze strany bezpečnosti si lze představit situaci, kdy uživatel aplikace zažádá o změnu hesla. Po chvilce se dostane na stránku, kde může vložit nové heslo. Jenže uživatel je nepředvídatelný a nevypočitatelný. Změnu hesla nedokončí a rozhodne se, že si chce procházet vtipné obrázky na podezřelé webové stránce. Tím se přenese celá URL změny hesla v hlavičce Referer a útočník na podezřelé webové stránce si ji může zapsat a použít, neboť uživatel celý proces změny hesla nedokončil a odkaz je stále platný.

Díky hlavičce Referrer-Policy je možné kontrolovat a omezovat množství informací, které se odesílá při odkazování na jiné stránky [30]. Správným nastavením lze ochránit uživatele před zjištěním identity na sociálních sítí pří sdílení odkazů nebo omezit informace pro analytické služby [30].

Hlavička může obsahovat pouze jednu z následujících direktiv [30]:

- no-referrer – hlavička Referer bude zcela vynechána (nepošle se).
- no-referrer-when-downgrade – odesílá se celá URL, pokud protokol zůstává stejný (HTTP → HTTP, HTTPS → HTTPS). V opačném případě no-referrer.
- origin – prohlížeč nastaví hodnotu pouze hlavní domény, zbytek URL je ignorován.
- strict-origin – stejné chování jako origin, avšak musí se shodovat úrovně protokolů.
- same-origin – celá URL je odeslána pouze na stejnou doménu. Jinak no-referrer.
- origin-when-cross-origin – na stejné doméně odesílá celou URL. Jinak posílá origin.
- strict-origin-when-cross-origin – přísnější varianta. Při odkazu na nižší úroveň zabezpečení (HTTPS → HTTP) bude hodnota stejná jako u no-referrer.
- unsafe-url – vždy se odešle celá URL. Nedoporučuje se.

```
# Usually the best choice, but everything depends on use of web site
Referrer-Policy: strict-origin-when-cross-origin
```

Zdroj. kód 20 – Nastavení hlavičky Referrer-Policy

4.2.4 Feature Policy

Pomocí této hlavičky je možné ovlivňovat funkce webového prohlížeče. Z pohledu bezpečnosti je možné například zakázat takové funkce prohlížeče, které by mohli být zneužity škodlivým kódem třetí strany [30]. Nicméně tato hlavička je stále ve vývoji a je experimentální.

4.2.5 Expect-CT

Jedná se o reportovací hlavičku, která umožňuje kontrolovat certifikát webových stránek. Poskytuje kontrolu nad tím, jak je SSL certifikát vyhodnocován v Certificate Transparency (kapitola 4.1.2). Umožňuje prohlížečům odesílat reporty, když mají problémy s ověřením certifikátu webových stránek [30]. Hlavička je odesílána pouze přes šifrované spojení HTTPS [33].

V hlavičce se nastavují tři direktivy:

- max-age – nastavuje maximální čas pro uchování hlavičky v mezipaměti prohlížeče.
- report-uri – URI kam se mají posílat reporty v případě, když CT vrátí chybu.
- enforce – zda má prohlížeč doporučení CT dodržovat.

```
Expect-CT: enforce, max-age=2592000, report-uri="https://www.example.com/report"
```

Zdroj. kód 21 – Nastavení hlavičky Expect-CT

4.2.6 X-Frame-Options

Hlavička zakazuje vkládání webové stránky či jeho části do jiných webových stránek při použití HTML značek <*frame*>, <*iframe*> a <*object*> [30]. Tím lze uživatele chránit proti útoku Clickjacking (kapitola 3.4). S touto myšlenkou přišel Microsoft v roce 2009 a implementoval ji do svých webových prohlížečů Internet Explorer. Dnes je podporována všemi moderními prohlížeči.

Lze použít následující direktivy [30]:

- DENY – jakékoliv vkládání je zakázáno.
- SAMEORIGIN – povoleno vkládání pouze v rámci domény.
- ALLOW-FROM – vkládání zakázáno, kromě zvolené domény.

Nevýhodou direktivy ALLOW-FROM je, že lze uvést pouze jednu doménu, což v některých situací nemusí vyhovovat. I z tohoto důvodu se doporučuje využít možnosti hlavičky CSP a tuto hlavičku použít spíše jako podpůrnou, pokud to situace dovoluje.

```
# Injection forbidden
X-Frame-Options: DENY

# Injection only at the same domain
X-Frame-Options: SAMEORIGIN

# Injection fobidden, given URI is exception
X-Frame-Options: ALLOW-FROM https://www.example.com
```

Zdroj. kód 22 – Možnosti nastavení hlavičky X-Frame-Options

4.2.7 X-Content-Type-Options

Mezi výchozí chování webových prohlížečů patří procházení odkazovaných souborů – tzv. content-sniffing a kontrola MIME typů [30]. Když například webová aplikace nastaví chybný MIME typ pro soubor, webový prohlížeč neví, jak ho zpracovat. Začne tedy soubor procházet a zkusí sám odhadnout správný MIME typ. Pokud prohlížeč najde při procházení souboru nějaký script, pokusí se ho spustit. To otevírá možnosti útočníkovi, neboť je možné vložit škodlivý script například do obrázku.

Použitím této hlavičky prohlížeči sdělujeme, že nemá procházet obsah souboru a odhadovat jeho MIME typ, a že se má řídit pouze informacemi ze serveru [30].

Tato hlavička disponuje pouze jedinou direktivou – nosniff. Je podporována ve všech moderních prohlížečích, ale konkrétní chování se může lišit dle implementace prohlížeče [30].

```
X-Content-Type-Options: nosniff
```

Zdroj. kód 23 – Nastavení hlavičky X-Content-Type-Options

4.2.8 X-XSS-Protection

V prohlížečích Internet Explorer (verze 8 a vyšší), Edge, Chrome a Safari lze použít bezpečnostní hlavičku X-XSS-Protection, která konfiguruje XSS filtry zabudované v prohlížeči [30]. Tyto filtry jsou schopny detekovat útok typu Reflected XSS (kapitola 3.2.1). V případě detekce prohlížeč v závislosti na nastavení nedovolí stránku zobrazit, nebo se pokusí odstranit škodlivou část stránky.

Bohužel, tyto filtry ne vždy fungují správně [34]. Občas bezpečný důležitý script pro chod webové stránky označí za nebezpečný. Taktéž se ukázalo, že efektivnější řešení nabízí hlavička CSP. Proto se vývojáři výše zmíněných webových prohlížečů rozhodli vývoj a podporu do budoucna ukončit [34]. Dnes, pokud k tomu nemáme důvod, se nedoporučuje tuto hlavičku používat (vypnout ji).

```
# Filter is disabled
X-XSS-Protection: 0;

# Filter is enabled
X-XSS-Protection: 1;

# Filter is enabled, in case of detection page is blocked
X-XSS-Protection: 1; mode=block

# Filter is enabled, incident is reported to given URI
X-XSS-Protection: 1; report=<reporting-uri>
```

Zdroj. kód 24 – Možnosti nastavení hlavičky X-XSS-Protection

4.2.9 X-Download-Options

Hlavička byla představena v roce 2008 a platí pouze pro webové prohlížeče Internet Explorer 8 a vyšší [30]. Pomocí ní přimějeme uživatele, aby soubor neotevíral přímo na stránkách, ale aby ho nejdříve uložil na počítači a poté jej teprve otevřel či spustil [30].

To přináší hned dvě výhody. Zaprvé, soubor může být zkontrolován antivirovým programem. Druhou výhodou je, že soubor nebude spuštěn v kontextu webové stránky, a tudíž případný vložený škodlivý skript v souboru bude neúčinný.

```
X-Download-Options: noopen
```

Zdroj. kód 25 – Nastavení hlavičky X-Download-Options

4.2.10 X-Powered-By

Hlavička X-Powered-By je nestandardizovaná informativní hlavička, využívána systémy a aplikacemi [30]. Nepatří mezi běžné bezpečnostní hlavičky, nicméně ji lze požadovat za bezpečnostní riziko. Hlavička totiž sděluje, na jakém systému či frameworku aplikace běží [30]. To je informace, která potencionálnímu útočníkovi ulehčuje práci a útok. Pokud neexistuje nějaký oprávnění důvod tuto informaci sdělovat, doporučuje se hodnotu hlavičky změnit na nečitelnou či hlavičku úplně zakázat.

```
# Setting unspecific value
X-Powered-By: "NONE"

# Unsetting header in .htaccess file of Apacher server
Header unset X-Powered-By
```

Zdroj. kód 26 – Nastavení hlavičky X-Powered-By

4.2.11 Server

HTTP hlavička Server je skoro stejná jako hlavička X-Powered-By. S tím rozdílem, že udává informace o typu webového serveru, který webový požadavek zpracoval [35]. Z pohledu bezpečnosti opět platí to, co u hlavičky X-Powered-By. Pokud není nutné tuto informaci sdělovat, nečiňme tak.

```
# Setting unspecific value
Server: "NONE"
```

Zdroj. kód 27 – Doporučené nastavení hlavičky Server

ZÁVĚR

Kryptografie je věda, která využívá aplikaci matematických metod k zajištění autentičnosti a důvěrnosti zpráv. Díky ní lze komunikaci šifrovat, či ověřovat původce zprávy. Kryptografie stále více získává na významu. Tento fakt je ještě umocněn čím dál vyšší poptávkou anonymity a ochrany soukromí ze strany uživatelů internetu.

S rostoucím počtem aplikací a služeb na internetu je taktéž nutné rozlišovat uživatele a jejich pravomoci. S tím jsou spojeny procesy autentizace a autorizace. Autentizace slouží k ověření identity. Zato pomocí autorizace zjistíme pravomoci této identity. Možností a technologií, jak tyto procesy implementovat, je dnes veliké množství, přičemž každá má své uplatnění ve specifických situací.

Útoků na webové aplikace je nespočetně a každým dnem přibývají. Organizace OWASP vývoj těchto útoků sleduje a monitoruje. Zdarma poskytuje edukaci a trénování moderní kyberbezpečnosti. Doporučuje využívání moderních frameworků, které vývoj webových aplikací urychlují a snaží se automatizovat zabezpečení většiny bezpečnostních rizik. Na bezpečnost musíme myslet i při komunikaci s klientem. Šifrované spojení pomocí protokolu HTTPS je dnes jedním z nejdůležitějších pilířů v oblasti kyberbezpečnosti.

Tato práce reflektovala pouze vybrané problematiky. Při budování webové aplikace nesmíme zapomenou na zabezpečení a správu operačního systému, webového serveru a databáze, které jsou dnes nedílnou součástí webových aplikací. Ty se také mohou stát obětí cíleného kybernetického útoku. Proto je nutné je spravovat a aktualizovat. Stejně tak se nesmíme podcenit fyzickou bezpečnost serverů. Pokud by útočník získal fyzický kontakt se serverem, veškerá ostatní opatření by přišla vniveč.

Bezpečnost je míra, nikoliv vlastnost. Tato míra je tvořena tématy a obory, které na sobě závisí a zároveň se doplňují. Logika aplikací a jejich zabezpečení se musí dělat takovým způsobem, které je odolné vůči nevypočitatelnému uživateli. Nelze předpokládat, že uživatel udělá vždy to, co má. Koneckonců, drtivá většina bezpečnostních incidentů vzniká právě kvůli uživatelům či zaměstnancům, kteří jsou neinformovaní, nedisciplinovaní, či je jim bezpečnost zcela lhostejná.

Na závěr je dobré si uvědomit, že nic není na 100 % bezpečné. Pokud někdo něco takového zaručuje, dává slib, který nelze dodržet. Vysoce motivovaní jedinci jsou schopni prolomit zabezpečení například bank či korporací, které investují do bezpečnosti nemalé peníze.

POUŽITÁ LITERATURA

[1] WU, Hanqing a Liz ZHAO. Web security: A WhiteHat Perspective. CRC Press, 2015. ISBN 978-14-6659-262-9

[2] SNYDER, Chris a Michael SOUTHWELL. Pro PHP Security. Apress, 2006. ISBN 978-14-3020-057-4

[3] BURDA, Karel. Úvod do kryptografie. Brno: Akademické nakladatelství CERM, 2015. ISBN 978-80-7204-925-7

[4] Úvod do kryptologie [online]. [cit. 2020-01-15]. Dostupné z: https://is.mendelu.cz/eknihovna/opory/zobraz_cast.pl?cast=7021

[5] Sessions | Nette Docs [online]. [cit. 2020-02-20]. Dostupné z: https://doc.nette.org/cs/3.0/sessions

[6] JSON Web Token Introduction – jwt.io [online]. [cit. 2020-02-27]. Dostupné z: https://jwt.io/introduction/

[7] OAuth 2.0 Authorization Framework [online]. [cit. 2020-03-06]. Dostupné z: https://auth0.com/docs/protocols/oauth2

[8] Vícefaktorová autentizace | AMI Praha a.s. [online]. [cit. 2020-02-18]. Dostupné z: https://www.ami.cz/publikujeme/blog/vicefaktorova-autentizace

[9] Bezpečnost hesla – Matematika.cz [online]. [cit. 2020-03-26]. Dostupné z: https://matematika.cz/bezpecnost-hesla

[10] The best password managers for 2020 and how to use them [online]. [cit. 2020-03-10]. Dostupné z: https://www.cnet.com/news/best-password-managers-for-2020/

[11] About the OWASP Foundation [online]. [cit. 2020-01-24]. Dostupné z: https://owasp.org/about/

[12] OWASP Top Ten [online]. [cit. 2020-01-23]. Dostupné z: https://owasp.org/www-project-top-ten/

[13] What is cross-site scripting (XSS) and how to prevent it? [online]. [cit. 2020-01-24]. Dostupné z: https://portswigger.net/web-security/cross-site-scripting

[14] Htmlspecialchars – Český PHP manuál [online]. [cit. 2020-01-27]. Dostupné z: https://www.itnetwork.cz/php/manual/retezce/funkce-htmlspecialchars-cesky-php-manual

[15] Latte – nejbezpečnější & opravdu intuitivní šablony pro PHP [online]. [cit. 2020-02-02]. Dostupné z: https://latte.nette.org/cs/

[16] Home – Twig – The flexible, fast, and secure PHP template engine [online]. [cit. 2020-02-02]. Dostupné z: https://twig.symfony.com/

[17] What is CSRF ? Tutorial & Examples | Web Security Academy [online]. [cit. 2020-03-19]. Dostupné z: https://portswigger.net/web-security/csrf

[18] PHP triky – Cross-Site Request Forgery [online]. [cit. 2020-03-19].
Dostupné z: https://php.vrana.cz/cross-site-request-forgery.php

[19] What is Confused deputy problem? [online]. [cit. 2020-03-19].
Dostupné z: https://www.thesecuritybuddy.com/vulnerabilities/what-is-confused-deputy-problem/

[20] Same-origin policy – Web security | MDN [online]. [cit. 2020-03-19].
Dostupné z: https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy

[21] What is Clickjacking? Tutorial & Examples | Web Security Academy [online].
[cit. 2020-03-16]. Dostupné z: https://portswigger.net/web-security/clickjacking

[22] NoScript – Javascript/Java/Flash blocker for a saver Firefox [online]. [cit. 2020-03-16].
Dostupné z: https://noscript.net/

[23] What is Session Hijacking? | Netsparker [online]. [cit. 2020-03-17].
Dostupné z: https://www.netsparker.com/blog/web-security/session-hijacking/

[24] Draft-foudil-securitytxt-09 - A File Format to Aid in Security Vulnerability Disclosure [online].
[cit. 2020-03-23]. Dostupné z: https://tools.ietf.org/html/draft-foudil-securitytxt-09

[25] RFC 5785 - Defining Well-Known Uniform Resource Identifiers (URIs) [online].
[cit. 2020-03-23]. Dostupné z: https://tools.ietf.org/html/rfc5785

[26] ŠPAČEK, Michal. HTTPS pro vývojáře a správce.
Dostupné z: https://www.michalspacek.cz/skoleni/https-pro-vyvojare-a-spravce

[27] Mozilla SSL Configuration Generator [online]. [cit. 2020-04-01].
Dostupné z: https://ssl-config.mozilla.org/

[28] SSL Server Test (Powered by Qualys SSL Labs) [online]. [cit. 2020-04-01].
Dostupné z: https://www.ssllabs.com/ssltest/

[29] HTTPS Everywhere | Electronic Frontier Foundation [online]. [cit. 2020-04-01].
Dostupné z: https://www.eff.org/https-everywhere

[30] Security Headers | Informace, návody a nastavení HTTP bezpečnostních hlaviček [online].
[cit. 2020-03-25]. Dostupné z: https://securityheaders.cz

[31] HSTS Preload List Submission [online]. [cit. 2020-03-27].
Dostupné z: https://hstspreload.org/

[32] Referer – HTTP | MDN [online]. [cit. 2020-03-26].
Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Referer

[33] Expect-CT – HTTP | MDN [online]. [cit. 2020-04-01].
Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Expect-CT

[34] X-XSS-Protection – HTTP | MDN [online]. [cit. 2020-03-25].
Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-XSS-Protection

[35] Server – HTTP | MDN [online]. [cit. 2020-03-25].
Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Server

[36] What really is the difference between session and token-based authentication [online]. [cit. 2020-02-20]. Dostupný na WWW: https://dev.to/thecodearcher/what-really-is-the-difference-between-session-and-token-based-authentication-2o39

[37] An Introduction to OAuth 2 | DigitalOcean [online]. [cit. 2020-03-06]. Dostupný na WWW: https://assets.digitalocean.com/articles/oauth/abstract_flow.png

[38] Session Hijacking Attack and Prevention | Security Testing [online]. [cit. 2020-03-18]. Dostupný na WWW: https://prashantshakti.wixsite.com/security/post/sessionhijacking

[39] What is CSRF ? Tutorial & Examples | Web Security Academy [online]. [cit. 2020-03-19]. Dostupný na WWW: https://portswigger.net/web-security/images/cross-site%20request%20forgery.svg

[40] What is an SSL Stripping Attack – Explained by SSL Experts [online]. [cit. 2020-03-31]. Dostupný na WWW: https://www.rapidsslonline.com/ssl/what-is-ssl-stripping-attack/