FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Modernizace podpory protokolů SSL/TLS v Privoxy
Student:	Bc. Václav Švec
Vedoucí:	Ing. Josef Kokeš
Studijní program:	Informatika
Studijní obor:	Počítačová bezpečnost
Katedra:	Katedra informační bezpečnosti
Platnost zadání:	Do konce letního semestru 2020/21

Pokyny pro vypracování

1) Proveďte rešerši vývoje šifrovaného HTTP v aktuálních prohlížečích za poslední tři roky. Zohledněte i změny v používaných certifikátech.
2) Zhodnoťte, jak tento vývoj ovlivňuje problematiku filtrování HTTPS pomocí proxy serveru. Vyjděte ze své předchozí implementace podpory SSL/TLS do Privoxy.
3) Navrhněte úpravy do Privoxy pro modernizaci jeho SSL/TLS podpory a zajištění jeho funkcionalit při filtrování komunikace moderních prohlížečů.
4) Analyzujte další uživatelské a programátorské nedostatky předchozí verze podpory SSL/TLS a navrhněte jejich řešení.
5) Proveďte takové úpravy v Privoxy, které popsané nedostatky vyřeší.
6) Otestujte původní i modernizovanou verzi Privoxy v aktuálně používaných prohlížečích.
7) Diskutujte dosažené výsledky.

Seznam odborné literatury

Dodá vedoucí práce.

prof. Ing. Róbert Lórencz, CSc.
vedoucí katedry

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

V Praze dne 31. ledna 2020

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Diplomová práce

Modernizace podpory protokolů SSL/TLS v Privoxy

Bc. Václav Švec

Katedra informační bezpečnosti
Vedoucí práce: Ing. Josef Kokeš

18. května 2020

Poděkování

Rád bych poděkoval Ing. Josefu Kokešovi za jeho cenné rady, podnětné připomínky a velmi vstřícný a obětavý přístup při vedení této práce.

Prohlášení

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

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo"), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 18. května 2020 .....

0.0.1 Odkaz na tuto práci

Švec, Václav. *Modernizace podpory protokolů SSL/TLS v Privoxy.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2020.

Abstrakt

Práce reaguje na vývoj šifrovaného HTTP v moderních webových prohlížečích během poslední tří let. Zejména se zaměřuje na změny, které se týkají proxy serverů a webových certifikátů a hodnotí jejich dopady na filtrování HTTPS pomocí proxy serverů. Součástí práce je i návrh a implementace úprav, které rozšiřují předchozí implementaci podpory SSL/TLS do proxy serveru Privoxy. Tyto úpravy umožňují filtrování veškeré komunikace moderních webových prohlížečů a zároveň kladou důraz na zachování dostatečného výkonu proxy serveru. Dosažené výsledky jsou testovány a zhodnoceny v porovnání s předchozími verzemi Proxy serveru Privoxy. Zároveň jsou doplněny o další vhodná vylepšení.

Klíčová slova webový prohlížeč, proxy server, filtrování, Privoxy, TLS, HTTPS

Abstract

The thesis reacts to the development of encrypted HTTP in modern web browsers over the last three years. It focuses particularly on changes that affect proxy servers and web certificates and evaluates their impact on HTTPS filtering using proxy servers. The thesis also includes the design and implementation of modifications that extend the previous implementation of SSL/TLS support to the Privoxy proxy server. These modifications allow filtering of all communication of modern web browsers and at the same time emphasize the maintenance of sufficient proxy server performance. The achieved results are tested and evaluated in comparison with previous versions of the Privoxy proxy server. At the same time, they are supplemented by other suitable improvements.

Keywords web browser, proxy server, filtering, Privoxy, TLS, HTTPS

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Technologie pro přenos dat mezi dvěma koncovými zařízeními se neustále vyvíjí od prvních experimentálních sítí až po nejnovější verze protokolu HTTP. V průběhu tohoto vývoje vznikly webové prohlížeče jako nástroje umožňující jednoduší a efektivnější získání a zobrazení dat. Moderní webové prohlížeče reagují na vývoj nových technologií a mnohdy tento vývoj i předurčují. Změnám se poté musí přizpůsobit i mezilehlá zařízení, jako jsou například proxy servery. Jen tak mohou uživatelům zaručit bezproblémovou funkčnost při použití moderních webových prohlížečů.

Proxy servery rozšiřují možnosti uživatelů například o filtrování veškeré přenášené komunikace. To umožňuje odstranění nežádoucího obsahu, jako jsou například reklamy. Cílené filtrování komunikace ovšem může sloužit i pro zvýšení bezpečnosti uživatele. K tomu slouží například blokování požadavků na potencionálně nebezpečné skripty, či změna obsahu požadavků tak, aby byla ztížena identifikace jejich odesilatele.

Cílem této diplomové práce je analýza moderních webových prohlížečů z hlediska vývoje šifrovaného HTTP během posledních tří let. Zároveň se práce věnuje souvisejícím změnám v používaných webových certifikátech, které mohou být během navazování šifrovaného HTTP spojení využity k autentizaci protistrany. Na základě této analýzy a analýzy předchozí verze podpory TLS v proxy serveru Privoxy jsou následně navržena a implementována vhodná rozšíření tohoto proxy serveru. Ta umožňují filtrovat veškerou HTTPS komunikaci přenášenou přes proxy server, a to včetně HTTP hlaviček. Zároveň tato rozšíření zohledňují vývoj aktuálních webových prohlížečů. Díky tomu je zaručena kompatibilita výsledného řešení s těmito prohlížeči.

Implementační část práce navazuje na výstupy bakalářské práce[1], která poskytuje rozšíření umožňující částečně filtrovat TLS komunikaci v proxy severu Privoxy. Proto se tato práce zaměřuje i na uživatelské a programátorské nedostatky předchozích verzí proxy serveru, pro které jsou navržena a implementována nová vhodnější řešení. Analýza nedostatků se pak zaměřuje

zejména na zjednodušení zdrojového kódu proxy serveru tak, aby byl jednodušeji spravovatelný a rozšiřovatelný. Dále jsou analyzovány možnosti pro zajištění dostatečného výkonu pro praktické využívání proxy serveru. Toho je dosaženo díky implementaci nových podpůrných mechanismů.

Součástí práce je i testování dosažených výsledků a vyhodnocení těchto testů, a to z pohledu funkčnosti, bezpečnosti a výkonnosti. Tyto výsledky jsou zároveň srovnány s výsledky moderních webových prohlížečů, případně s předchozími verzemi proxy serveru Privoxy. Naznačena jsou i vhodná budoucí rozšíření proxy serveru.

Kapitola 1

Vývoj šifrovaného HTTP

Protokol HTTP (HyperText Transfer Protocol) vytvořil v letech 1989 až 1991 Tim Berners-Lee a jeho tým. Jedná se o základní protokol pro komunikaci klienta s WWW (World Wide Web) servery, který umožňuje přenášet nejen hypertextové dokumenty, ale i jiné typy souborů. Od svého vzniku až do současnosti prošel řadou změn. Tato kapitola se věnuje vývoji tohoto protokolu se zaměřením na jeho šifrovanou verzi, tedy HTTP secure (HTTPS).[2]

1.1 První verze protokolu a její vylepšení

První verze protokolu byla později označena HTTP/0.9. Celý požadavek se v této verzi skládal pouze z jedné řádky s jedinou možnou metodou `GET`. Odpověď obsahovala pouze vyžádaný soubor. Kvůli svým limitům byla tato verze brzy rozšířena o stavové kódy v každé odpovědi serveru. Ty umožnily webovému prohlížeči detekovat chyby při zpracování požadavků a přizpůsobit jim své chování. Jedním z dalších rozšíření byla definice HTTP hlavičky, a to jak v požadavku klienta, tak v odpovědi serveru. Ta umožnila přenášet metadata, a tím učinila protokol flexibilním a rozšiřitelným. Takto rozšířená verze je označována jako HTTP/1.0.[2]

Následující verze HTTP/1.1 je první standardizovanou verzí protokolu. Byla publikována v roce 1997 jen několik měsíců po verzi HTTP/1.0. Z řady vylepšení je důležité zejména odstranění nedostatků při opakovaném používání spojení. Díky tomu může být uspořen čas na jeho opětovné navazování, pokud klient žádá více dokumentů od jednoho serveru. Je umožněno zaslat druhý požadavek na server před přijetím úplné odpovědi na požadavek předchozí. Tím je snížena latence komunikace. Díky parametru `Host` v hlavičce požadavku, který je v této verzi protokolu již povinný, je možné provozovat několik různých domén pod stejnou IP adresou. Přidání dalších parametrů do hlaviček umožňuje vyjednávání o vlastnostech žádaných dokumentů, jako je např. jazyk či kódování. Protokol HTTP/1.1 byl díky své snadné rozšiřitelnosti vylepšen dvěma revizemi a je dodnes využívaný.[2]

1.2 HTTP/2

S rostoucím objemem přenášených dat a narůstající složitostí webových stránek výrazně vzrostl i počet HTTP požadavků. Lze využít paralelní spojení, což ovšem přináší značnou režii a složitost komunikace. V roce 2010 představila společnost Google alternativní způsob přenosu dat mezi klientem a serverem pomocí experimentálního protokolu SPDY, který se zaměřuje právě na tyto problémy. Hlavní rozdíly oproti HTTP/1.1 jsou:[2][3]

- Na rozdíl od předchozích verzí, které byly textové, je HTTP/2 binární. To umožňuje jeho jednoduší a rychlejší zpracování.
- Paralelní požadavky mohou být přenášeny přes jedno spojení bez ohledu na jejich pořadí. Zpoždění jednoho dotazu tak neblokuje ostatní. Stačí tedy navázat pouze jedno TCP spojení mezi klientem a serverem.
- Dochází ke kompresi hlaviček, které jsou často podobné napříč jednotlivými dotazy. Odstraňují se tak duplicity a snižuje režie.
- Pomocí mechanismu Server push je serveru umožněno odeslat data klientovi dříve, než si je vyžádá.
- Příjemce může využít mechanismy pro řízení toku dat od odesilatele a zároveň může ovlivnit i některé parametry protokolu. Díky tomu dokáže chránit své omezené prostředky.

Protokol vycházející z SPDY byl publikován v roce 2015 jako HTTP/2 a nyní je používán přibližně 43 % webových stránek. Je tedy provozován paralelně s předchozími verzemi protokolu.[4] Původně se předpokládalo jeho používání pouze přes zabezpečené spojení. To nakonec není striktně vyžadováno, ale některé webové prohlížeče nešifrované HTTP/2 spojení neumožňují.[3]

1.3 HTTPS

HTTP protokol využívá pro přenos dat protokol TCP/IP. Zásadní změna ovšem nastala v roce 1994, kdy společnost Netscape Communications rozšířila protokol o šifrovanou vrstvu, tedy o protokol SSL. Tím zajistila důvěrnost přenášených dat mezi klientem a serverem a zároveň umožnila kontrolovat i integritu a autentičnost těchto dat. Vznikl tak protokol HTTPS (HyperText Transfer Protocol Secure). Následně byl protokol SSL nahrazen protokolem TLS. Tím se odstranila řada původních zranitelností.[2]

1.4 Aktuální vývoj ve webových prohlížečích

I v současnosti se protokoly HTTP a HTTPS neustále vyvíjí. Webové prohlížeče i proxy servery tak musí neustále reagovat na tyto změny. Tato sekce popisuje změny ve webových prohlížecích od začátku roku 2017, které se týkají zejména protokolu HTTPS. Následuje tabulka srovnávající aktuálně široce rozšířené webové prohlížeče podle implementace těchto rozšíření.

1.4.1 RSA-PSS pro TLS

Řada kryptografických metod se spoléhá na předpoklad, že pokud veřejně zdokumentované a dobře známé algoritmy byly přezkoumány velkým počtem odborníků bez jakýchkoli průlomů, je tento algoritmus bezpečný. V minulosti se ale ukázalo, že i algoritmy které byly považovány za bezpečné, byly po čase prolomeny (MD5, SHA-1 a RSA 512/768 bit). Je ale žádoucí poskytnout tzv. prokazatelné zabezpečení. To nemůže označit celý kryptografický systém za bezpečný, ale může prohlásit, že bezpečnost systému souvisí s bezpečností základního problému.[5]

V roce 1996 navrhli Mihir Bellare a Phillip Rogaway dvě prokazatelně bezpečná schémata pro RSA: Probabilistic Signature Scheme (PSS) pro podepisování a Optimal Asymmetric Encryption Padding (OAEP) pro šifrování. Bezpečnost těchto schémat tedy přímo souvisí s bezpečností RSA a použitou hashovací funkcí.[5]

TLS 1.3 se od TLS 1.2 liší mimo jiné i ve změně RSA padding na používání RSASSA-PSS.[6] V rámci příprav na podporu protokolu TLS 1.3 tak některé webové prohlížeče s předstihem implementovaly PSS algoritmy pro podepisování i při využití protokolu TLS 1.2.[7]

1.4.2 Odstranění CBC módu u ECDSA TLS

CBC mód pro TLS je problematický a je velmi obtížné ho implementovat bezpečně. Ačkoliv je šifrovací mód CBC velmi rozšířený pro RSA, pro ECDSA (Elliptic Curve Digital Signature Algorithm) se prakticky nevyužívá. Proto se některé webové prohlížeče rozhodly z podporovaných šifrových sad TLS odebrat ECDHE_ECDSA_WITH_AES_128_CBC_SHA a ECDHE_ECDSA_WITH-_AES_256_CBC_SHA.[7]

1.4.3 Odebrání TLS 1.2 ECDSA s SHA-1 a SHA-512

TLS handshake se pro verzi TLS 1.2 skládá z řady kroků. Klient iniciující handshake zasílá na server tzv. `Client Hello Message`. Kromě náhodných bytů, session ID (pokud již bylo spojení dříve navázáno) a seznamu šifrových sad může obsahovat i seznam požadovaných rozšíření. Jedním z nich je i „signature_algorithms“. Díky němu může klient říci, které páry algoritmů (pro

podpis a pro hash) mohou být použity v digitálních podpisech, tedy které algoritmy u digitálních podpisů bude akceptovat.[8]

Pokud server podporuje některou ze šifrových sad přijatých od klienta, odpovídá na jeho zprávu. Ta obsahuje náhodné číslo, verzi šifrové sady podle své preference, session ID a seznam rozšíření, která klient vyžaduje a zároveň mu je může server poskytnout. Pokud server nepodporuje všechna rozšíření vyžadovaná klientem, musí i tak spojení akceptovat. Klient se pak může rozhodnout, jestli chce ve spojení pokračovat.[8]

Aby byla snížena závislost webových prohlížečů na algoritmu SHA-1, a aby se webové prohlížeče připravily na jiné zacházení s ECDSA v TLS 1.3, odstraňují algoritmy ECDSA s SHA-1 a ECDSA s SHA-512 z rozšíření `signature_algorithms` zasílaných na server. Pro protokol ECDSA byly obvykle ponechány pouze SHA-256 a SHA-384.[7]

1.4.4 Ukončení důvěry pro SHA-1 certifikáty

Hashovací funkce SHA-1 byla oficiálně označena za zastaralou americkým National Institute of Standards and Technology (NIST) v roce 2011, a to pro bezpečnostní slabiny prokázané v různých analýzách a teoretických útocích. Přesto byla používána ještě v roce 2017 pro podepisování dokumentů, TLS certifikátů i v dalších aplikacích, jako je například Git. V roce 2017 představila skupina odborníků útok, kdy se jim podařilo vytvořit dva různé validní PDF dokumenty se stejným SHA-1 otiskem. Jedná se tedy o útok nalezením kolize, tzn. nalezení dvou různých vstupů, pro které SHA-1 vygeneruje stejný otisk, který není předem definovaný. To vše se podařilo přibližně 100 000 krát rychleji než při použití útoku hrubou silou. [9]

V reakci na publikování tohoto útoku přestaly webové prohlížeče důvěřovat certifikátům, které byly vytvořené právě pomocí SHA-1.

1.4.5 Odebrání nestandardizované verze ChaCha20-Poly1305

ChaCha20-Poly1305 je šifra typu Authenticated Encryption with Additional Data (AEAD). Šifry typu AEAD jsou speciální díky kombinaci šifrovacího algoritmu a Message Authentication Code (MAC) v jednom celku. Šifrovací algoritmy lze kombinovat s MAC i samostatně, ale i dva samostatně bezpečné prvky lze spojit do nebezpečné kombinace. Došlo například k prolomení některých dvojic díky použití stejného klíče pro šifrovací algoritmus i MAC (AES-CBC s CBC-MAC).[10]

V roce 2013 byla ve verzi 31 webového prohlížeče Chrome nasazena nová TLS šifrová sada ChaCha20-Poly1305. Ta byla později standardizována jako RFC 7539 a RFC 7905. Tato standardizovaná verze byla nasazena do prohlížeče Chrome v roce 2016, a tak mohla být o rok později odstraněna původní nestandardizovaná verze.[7] Další z hlavních webových prohlížečů zavedly až standardizovanou verzi protokolu, a tak nemusely předchozí verzi odebírat.

1.4.6 Cookies striktně přes HTTPS spojení

Parametrem `Set-Cookie` v hlavičce HTTP odpovědi může server požádat klienta, aby na své straně uložil data následující za tímto parametrem. Za daty pak může následovat i parametr `Secure` jako na příkladu níže:

```
Set-Cookie: <cookie-name>=<cookie-value>; Secure
```

Takto označené cookies smí klient odeslat v požadavku na sever pouze pokud je spojení se serverem zabezpečené.[11] Nicméně starší verze prohlížečů umožňovaly přidat a odebrat Secure cookies, i když byly příslušné pokyny přijaty přes nezabezpečené spojení. Nové aktualizace už ale znemožňují jakkoli ovlivňovat Secure cookies přes nezabezpečené spojení.[7]

1.4.7 Podpora shody `Common Name` v certifikátech

Aby klient předešel útokům typu Man In The Middle (MITM) na HTTPS spojení, musí porovnat hostname serveru s hostname uvedeným v certifikátu, který od něj obdržel. RFC 2818 z roku 2000 říká, že pokud certifikát obsahuje rozšíření `Subject Alternative Name` typu `DNSName`, pak musí být hodnota tohoto rozšíření použita pro ověření identity serveru. V opačném případě musí být použito nejspecifičtější pole `Common Name` v poli `Subject`. Zároveň RFC dodává, že ačkoliv je ověřování identity pomocí pole `Common Name` v praxi používané, bylo již v roce 2000 zastaralé a certifikační autority (CA) by měly používat pole `DNSName`.[12] Webové prohlížeče tak postupně odstraňují podporu kontrolování identity serveru pomocí zastaralého parametru `Common Name` a přecházejí právě na doporučené rozšíření `Subject Alternative Name`.

1.4.8 Přihlašovací údaje v dotazech na subresource

Url adresa se skládá z řady částí, z nichž každá má svůj specifický účel. Jedna z nepovinných částí předcházející doménovému jménu slouží pro vložení přihlašovacích údajů. Od doménového jména je oddělena znakem „@“. Výsledná url pak může vypadat například takto:

```
http://ima_user:hunter2@example.com/yay.tiff
```

Nevýhoda této metody spočívá v tom, že data jsou přenášena jako prostý text. Zároveň může dojít ke zmatení uživatele, který může přihlašovací údaje považovat za doménové jméno. Skutečné doménové jméno pak může být maskováno kódováním. Některé webové prohlížeče se tak rozhodly pro zvýšení bezpečnosti omezit zasílání přihlašovacích údajů v rámci dotazů na subresources, tedy při načítání kaskádových stylů, skriptů nebo obrázků v rámci webové stránky. Dotazy na subresources obsahující přihlašovací údaje tak jsou prohlížečem blokovány, nebo je uživatel varován před touto skutečností.[7][13]

1.4.9 Certificate Transparency (CT)

Webové prohlížeče mohou detekovat škodlivé weby s neplatnými certifikáty. Současné kryptografické mechanismy však nejsou tak dobré v detekci škodlivých webů, pokud jsou jejich certifikáty vydány certifikační autoritou (CA) chybně, nebo pokud je vydala CA, která byla zkompromitována nebo je nečestná. V takových případech webový prohlížeč nevidí na certifikátu nic podezřelého, protože CA je z jeho pohledu důvěryhodná. Jedním z problémů je to, že neexistuje jednoduchý způsob, jak sledovat certifikáty v reálném čase. Podezřelé certifikáty nejsou obvykle detekovány a revokovány několik týdnů či měsíců. Navíc se tyto problémy opakují čím dál častěji. Jako příklad můžeme uvést CA DigiNotar, která byla napadena a útočníci dokázali vydávat libovolné falešné certifikáty.[14]

CT je služba pro monitorování systému certifikátů a jeho auditování. Snaží se zmírnit zmíněné problémy tím, že vydávání a existence certifikátů může být kontrolována kýmkoli, tedy vlastníky domén, certifikačními autoritami i uživateli domény. Klade si zejména následující cíle:[15]

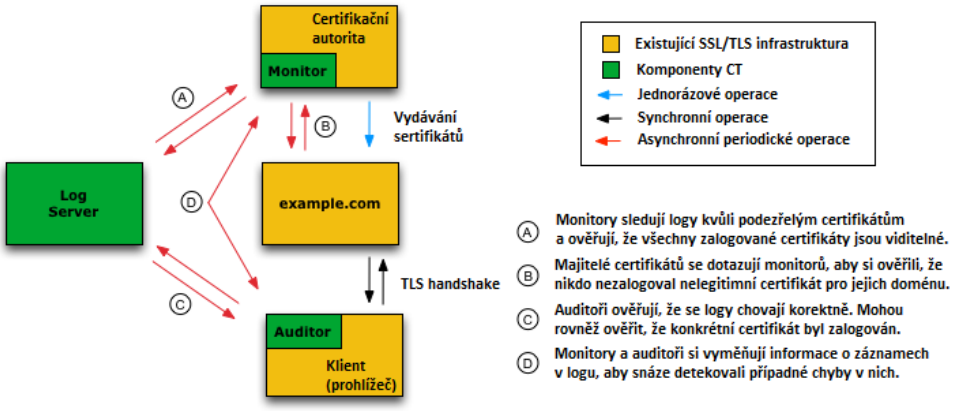
- Znemožnit nebo alespoň ztížit CA vydat certifikát pro doménu, aniž by o tom věděl vlastník domény.
- Umožnit komukoli odhalit, pokud byly certifikáty vydány CA omylem, chybně nebo se zlým úmyslem.
- Přimět klienty, aby odmítli uznat certifikáty, které se neobjevují v CT logu. Tím by byly CA donuceny přidávat do protokolů všechny vydané certifikáty.

Služba CT se skládá z několika základních částí.[15] Vzájemná komunikace mezi nimi je pak popsána na obrázku 1.1.

- Logy certifikátů

  Jedná se o jednoduché síťové služby, které umožňují komukoli zalogovat řetězec certifikátu. Logy lze pouze přidávat, nikoli měnit či odebírat. Toho je dosaženo pomocí tvz. Merkleho stromu. Jeho princip zároveň umožňuje efektivně odhalit, pokud se log pokusí odeslat různým klientům různé výstupy, či jiné nekorektní chování logu.

  Očekává se, že všechny CA budou zveřejňovat všechny své nově vydané certifikáty do jednoho nebo několika logů, případně že to bude provedeno majiteli certifikátů. Aby nedocházelo ke zbytečnému zahlcení logů, vyžaduje se, aby každý řetězec certifikátu měl jako kořenovou CA některou ze známých CA. Po zalogování řetězce certifikátu je navráceno podepsané časové razítko (Signed Certificate Timestamp, tj. SCT). K podpisu slouží soukromý klíč CT logu. SCT je příslib logu, že začlení přijatý řetězec certifikátu v definovaném časovém intervalu.



Obrázek 1.1: Komunikace jednotlivých prvků CT (upraveno podle [15])

- Monitory

  Monitory sledují logy a kontrolují, jestli se chovají korektně. Monitory musí přinejmenším kontrolovat každý nový záznam v logu, který sledují. Mohou také uchovávat kopie celých logů.

- Auditoři

  Auditoři berou jako vstup částečné informace z logu a ověřují, jestli jsou v souladu s dalšími částečnými informacemi, které mají.

- TLS klienti

  TLS klienti nejsou přímo klienti CT logů, ale získávají SCT jinými cestami, například v parametrech certifikátu, nebo na vyžádání od serveru při TLS handshake. Při validaci certifikátu (resp. navazování TLS spojení) by pak navíc měli ověřit podpis SCT, že je skutečně podepsaný validním CT logem, a že SCT byl opravdu vytvořený pro ověřovaný certifikát. Pokud jsou objeveny nesrovnalosti (např. timestamp SCT z budoucnosti), měl by klient spojení odmítnout. Jak má TLS klient získat veřejný klíč CT logu, ale RFC nespecifikuje.

Webové prohlížeče tak postupně začaly podporovat tuto službu. Jednou z konkrétně zavedených změn je například podpora `Expect-CT` v hlavičce serveru. Server nastavuje tento parametr, aby upozornil webový prohlížeč, že certifikáty serveru jsou zapsány v CT logu a prohlížeč by je měl kontrolovat i pomocí služeb CT.[7] Po úspěšném zavedení CT pak webový prohlížeč Chrome oznámil, že od dubna 2018 bude vyžadovat zapsání všech nově vydaných veřejně důvěryhodných certifikátů do CT logu.[16] Tím jsou CA nuceny vkládat veškeré nově vydané certifikáty do CT logu, jinak je tento webový prohlížeč označí za nedůvěryhodné.

1.4.10 Notifikační API jen přes HTTPS

Notifikační API dovoluje webovým stránkám a aplikacím zasílat notifikace, které se zobrazí mimo webový prohlížeč. Mohou tak zasílat informace uživateli, i když jsou ve stavu idle nebo běží na pozadí.[17] Webové prohlížeče rozhodují, kdo a za jakých podmínek může používat API pro vytváření notifikací. Některé se tak rozhodly omezit přístup skrz nezabezpečené HTTP spojení.[7]

1.4.11 TLS 1.3

TLS 1.3 je revize protokolu TLS, která byla definována v RFC 8446 v srpnu 2018. Webové prohlížeče rychle zareagovaly a brzy začaly podporovat tuto verzi protokolu. Při návrhu TLS 1.3 byl kladen důraz především na zvýšení výkonnosti a bezpečnosti protokolu. Některé z hlavních rozdílů a vylepšení oproti předchozí verzi TLS jsou vyjmenovány v následujícím seznamu:[6]

- Podporované algoritmy

  Seznam podporovaných symetrických šifrovacích algoritmů byl zkrácen o algoritmy, které jsou dnes považovány za zastaralé. Zachovány byly pouze algoritmy, které podporují AEAD. Tedy algoritmy, které kromě důvěrnosti přenášených dat poskytují i kontrolu integrity a autentičnosti těchto dat. Klient tak může zároveň ověřit původ dat a skutečnost, že data nebyla během přenosu neoprávněně upravena.[18]

- Zero round-trip time (0-RTT) mód

  Po navázání TLS 1.3 spojení mezi klientem a serverem zasílá server klientovi tzv. Pre-Shared Key (PSK). Při dalším navazování spojení může být tento klíč využit k ověření identity klienta a k zašifrování dat. PSK tak nahrazuje session ID a session tickets z předchozích verzí TLS. Zároveň může klient díky rozšíření `early_data` zaslat aplikační data už v první skupině zpráv na server. To značně urychlí opětovné vytváření spojení i přenos prvních dat.

- Šifrovaný handshake

  Nově přidaná zpráva typu `EncryptedExtensions` následuje vždy bezprostředně za zprávou `ServerHello` a je to první zpráva, která je šifrovaná. Zpráva odpovídá na seznam rozšíření vyžadovaných klientem. Všechny následující zprávy handshake jsou také šifrovány. Tím se zlepšilo zabezpečení handshake.

- Restrukturalizace handshake

  Byla změněna struktura stavového automatu pro handshake oproti předchozím verzím protokolu. Došlo také k redukci nadbytečných zpráv zasílaných během handshake. Tyto změny urychlují TLS handshake.

- Kryptografická vylepšení

  Nová verze přinesla také řadu kryptografických vylepšení. Jako příklad můžeme uvést přidání nových podpisových algoritmů jako jsou EdDSA. Naopak byly odebrány některé zastaralé algoritmy jako statické RSA a Diffie-Hellman.

1.4.12 Komprese certifikátů

Certifikáty často představují značnou část přenášených dat při TLS handshake. Proto by bylo velmi výhodné provést před jejich přenosem kompresi, a díky tomu snížit velikost přenášených dat během handshake. TLS 1.3 zavedlo šifrování certifikátů, díky kterému je možné implementovat kompresi certifikátů, aniž by to způsobovalo problémy na mezilehlých zařízeních. Řada z nich se totiž v praxi pokouší odchytávat přenášené certifikáty, provést jejich ověření a případně přerušit navazování spojení. Pokud by však místo očekávaného certifikátu tato zařízení odhalila na jeho místě pouze nesrozumitelná data komprimovaného certifikátu, mohla by přikročit k přerušení spojení. Protože jsou však certifikáty v TLS 1.3 šifrovány, nemůže k takovým problémům docházet.[19]

Webové prohlížeče tak podstupují úpravy, aby mohly dekomprimovat přijaté certifikáty, a následně s nimi zacházet stejně jako u starších protokolů. Z pohledu proxy serverů ale k žádným změnám nedochází, protože k dekomprimaci certifikátů by mělo docházet už v kryptografických knihovnách. Pokud ale proxy servery prováděly kontrolu přenášených certifikátů v rámci TLS handshake i při použití TLS tunelu, bude jim to šifrováním certifikátů v TLS 1.3 znemožněno.

1.4.13 AppCache pouze pro zabezpečené HTTP

AppCache je uložiště poskytované HTML 5, které umožňuje webovým aplikacím fungovat i bez připojení k Internetu. Do AppCache si aplikace může ukládat soubory, které jsou specifikovány v souboru `manifest`, který je uložený na webovém serveru a server ho zasílá klientovi. Taková funkcionalita ale značně rozšiřuje i možnosti útočníků. Do AppCache může být například uložen soubor s útokem typu Cross-Site Scripting (XSS), který se díky podvrženému manifestu uchová i při dalších načteních webové stránky. Proto webové prohlížeče omezují AppCache pouze pro připojení přes zabezpečené HTTP spojení. Postupně také přicházejí s proklamacemi, že v následujících verzích bude podpora AppCache zcela odstraněna.[20]

1.4.14 Úprava hodnot `User-Agent`

RFC 7231[21] říká, že atribut `User-Agent` HTTP hlavičky obsahující informace o systému klienta by neměl obsahovat žádné zbytečné detaily. Ty by

mohly posloužit útočníkům při hledání konkrétních slabin v systému klienta a zároveň zjednodušit identifikaci konkrétního klienta. Webové prohlížeče tak v souladu se zmíněným RFC v nových verzích odstraňují detaily, jako je například číslo buildu operačního systému.[7]

1.4.15 Token Binding

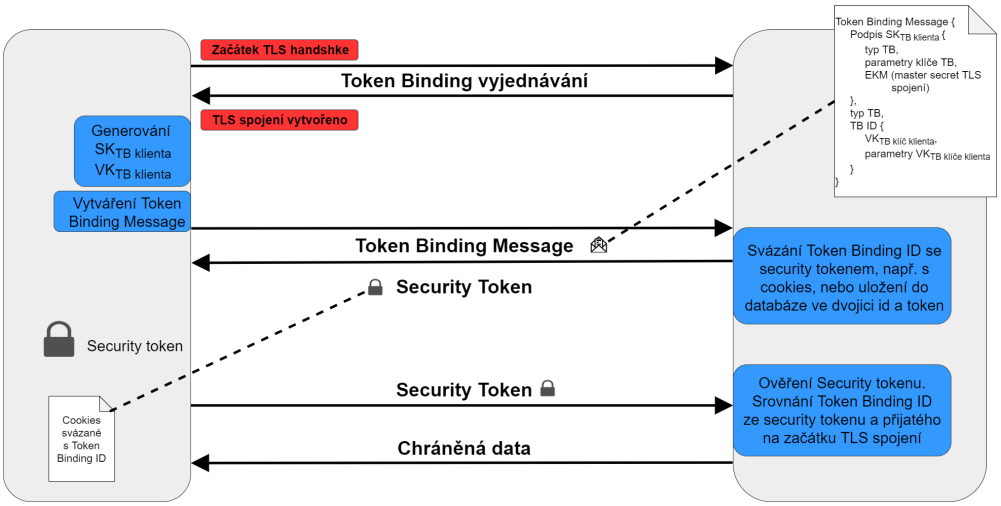
Servery generují tokeny (např. HTTP cookies), které pak slouží aplikacím k autentizaci při dalších přístupech k chráněným prostředkům. Toho mohou zneužívat útočníci tím, že získají tyto tokeny z komunikace nebo ze zařízení a zasílají je na server, čímž se vydávají za autentizované uživatele (tzv. replay útok). Protokol Token Binding se snaží zabránit takovým útokům pomocí kryptografického svázání aplikačních tokenů s TLS vrstvou. Při zaslání tokenu z jiného zabezpečeného spojení, než pro které byl token vytvořen, je tato skutečnost odhalena. Pro zjednodušení lze na protokol nahlížet z pohledu každé ze zúčastněných stran:[22]

1. Token Binding z pohledu klienta

   Klient vygeneruje dvojici soukromého a veřejného klíče pro každý server, ideálně s použitím bezpečného HW modulu jako je např. Trusted Platform Module (TPM). Serveru je zároveň prokázáno, že klient je vlastníkem odpovídajícího soukromého klíče. Jako důkaz slouží Token Binding Message, která obsahuje Exported Keying Material (EKM) z akutálně používaného TLS spojení podepsané soukromým klíčem klienta. EKM hodnota je odvozená od hodnoty TLS master secret, která je využívána k odvození klíčů TLS spojení. EKM tak označuje navázané TLS spojení. Tím se zabraňuje replay útokům, protože hodnota Token Binding Message je závislá na TLS spojení, ale zároveň je nutné, aby aplikace synchronizovala generování a ověřování Token Binding Message s TLS handshake.

   Token Binding Message slouží k prokázání vlastnictví privátního klíče klientem. Musí být poslána, pokud server s klientem úspěšně dojednali užití protokolu Token Binding během TLS handshake, a musí být poslána v první zprávě klienta, která obsahuje aplikační data. Případně může být i v následujících zprávách, aby bylo prokázáno vlastnictví i dalších privátních klíčů. Zpráva obsahuje sérii Token Binding struktur, z nichž každá obsahuje Token Binding identifikátor (ID), typ Token Binding a soukromým klíčem klienta podepsané zřetězení EKM, parametrů klíče a Token Binding typu.

   Token Binding ID jsou struktury obsahující identifikátor vyjednaných parametrů veřejného klíče, veřejný klíč klienta a jeho délku. Mají dlouhou životnost. To znamená, že se používají pro více TLS relací a spojení. Tomu by měly být přizpůsobeny i parametry klíče. Token Binding ID



Obrázek 1.2: Komunikace při použití protokolu Token Binding

také nikdy nesmí být přenášené přes nezabezpečené spojení a klient je může kdykoli zneplatnit.

2. Token Binding z pohledu serveru

   Po přijetí Token Binding Message od klienta musí server ověřit, jestli přijatý veřejný klíč klienta splňuje sjednané parametry. Zároveň server ověří podpis v této zprávě. V případě nesrovnalostí musí server Token Binding odmítnout. Pokud kontrola proběhne úspěšně, server předá Token Binding ID aplikaci na serveru a ta ho může použít pro vytvoření a ověřování security tokenu. Schéma komunikace je znázorněno na obrázku 1.2.

   TLS vrstvu lze svázat se security tokenem například vložením Token Binding ID nebo jeho hashe do security tokenu, případně ukládáním tokenů a k nim příslušejícím Token Binding ID. Způsob svázání volí aplikace. Po přijetí security tokenu od klienta server kontroluje Token Binding ID z tokenu a z TLS spojení s klientem. Pokud si neodpovídají, musí server token odmítnout. Pokud je přijat klasický a svázaný token, záleží na politikách aplikace, jestli bude dál zpracováván. Aby mohl útočník úspěšně získat a zneužít (replay attack) svázaný token, musel by použít soukromý klíč klienta, který si klient chrání.

Jako příklad využití Token Binding můžeme uvést protokol HTTPS. Poté co klient se serverem vyjednají použití protokolu Token Binding a naváží TLS spojení, posílá klient první HTTP požadavek, ve kterém musí uvést hlavičku

```
Sec-Token-Binding = EncodedTokenBindingMessage
```

obsahující Token Binding Message zakódovanou pomocí Base64url. Server po ověření Token Binding Message sváže Token Binding ID s cookies, které slouží jako security token. K resetování tokenu dojde, pokud klient smaže cookies z webového prohlížeče.[23]

Ačkoli Token Binding zlepšuje ochranu uživatelů, řada webových prohlížečů ho nikdy nezačala podporovat a jiné ho postupně odstraňují. Důvodem jsou vysoké náklady na zavedení a údržbu v porovnání se získanými přínosy. Jelikož se tato metoda nikdy široce nerozšířila, objevují se i problémy s kompatibilitou. Navíc existují alternativní způsoby k dosažení podobných výsledků, jako je příznak `HttpOnly` u cookies, či využívání klientských certifikátů.[24]

1.4.16 Zneplatnění certifikátů Symantec

V lednu 2017 bylo na veřejném fóru mozilla.dev.security.policy upozorněno na několik podezřelých certifikátů vydaných infrastrukturou veřejných klíčů (PKI) společnosti Symantec. Symantec provozuje řadu CA jako GeoTrust, RapidSSL a další. Tyto CA vydávaly řadu certifikátů, které nesplňovaly náležitosti průmyslových CA. Během následného vyšetřování bylo odhaleno, že Symantec svěřil důvěru několika organizacím, aby vydávaly certifikáty, aniž by zajistil náležitý a nezbytný dohled nad jejich počínáním. Navíc si společnost byla určitou dobu vědoma bezpečnostních nedostatků v těchto organizacích a nepodnikla žádná opatření. Nejednalo se o první bezpečnostní incident této společnosti. Naopak se jednalo o další z řady pochybení, a tak se webový prohlížeč Chrome rozhodl ukončit důvěru v PKI společnosti Symantec a postupně i v certifikáty, které vydala. Následně se k tomuto postupu připojily i další webové prohlížeče.[25]

1.4.17 Odebrání HTTP Public Key Pinning

HTTP Public Key Pinning (HPKP) mimo jiné rozšiřuje HTTP hlavičku odpovědi serveru o novou položku `Public-Key-Pins`. Ta umožňuje informovat webové prohlížeče o tom, jaké veřejné klíče by měl obsahovat řetězec certifikátu pro určitý hostname při dalších TLS spojeních. Povinný parametr `max-age` říká, kolik sekund po přijetí hlavičky od serveru je tento záznam platný. Parametr `pin` pak specifikuje hash Subject Public Key Information (SPKI) a algoritmus, který byl pro výpočet hashe použit. Pro případ, kdy by sever ztratil kontrolu nad svým privátním klíčem, je nutné zasílat alespoň jeden další záložní, který není serverem aktuálně používaný. Díky němu lze obnovit, vytvořit a verifikovat nové spojení při ztrátě privátního klíče. Nepovinný parametr `includeSubDomains` znamená, že kontrola klíče má být provedena i u všech subdomén dané domény. Další nepovinný parametr `report-uri` definuje URI, kam mají být hlášeny chyby při kontrole klíče. Hlavička odeslaná serverem pak může vypadat například takto:[26]

```
Public-Key-Pins:
      pin-sha256="E9CZ9INDbd+2eRQozYqqbQ2yXLVKB9+xcprMF+44U1g=";
      pin-sha256="LPJNul+wow4m6DsqxbninhsWHlwfp0JecwQzYpOLmCQ=";
      report-uri="http://example.com/pkp-report";
      max-age=2592000; includeSubDomains
```

Pokud je HPKP správně nastavené, umožňuje snižovat riziko útoků typu MITM i dalších, které se zaměřují na prolomení autentizace pomocí certifikátů. Při chybném nastavení HPKP ale může dojít k znepřístupnění určitých domén. HPKP je často používáno společně s HTTP Strict Transport Security (HSTS), které umožňuje oznámit webovému prohlížeči, že na určitý web má vždy přistupovat pomocí protokolu HTTPS. Společné používání obou protokolů ale není podmínkou.[26]

HPKP je mechanismus typu trust-on-first-use. To znamená, že při prvním připojení na server nemá klient informace potřebné k provedení kontroly pomocí HPKP a získává je právě z prvního připojení. Pokud tak je už při prvním připojení na server použit útok MITM, může si klient spojit doménu s podvrženým klíčem a po ukončení útoku odmítat pravý certifikát serveru. Zmírnit tento problém může seznam domén a jejich klíčů předinstalovaný v prohlížeči. Ten se tak nemusí spoléhat na první připojení k serveru. Do tohoto seznamu však nelze jednoduše zapsat libovolnou doménu. Jsou zde hlavně často navštěvované weby.[26]

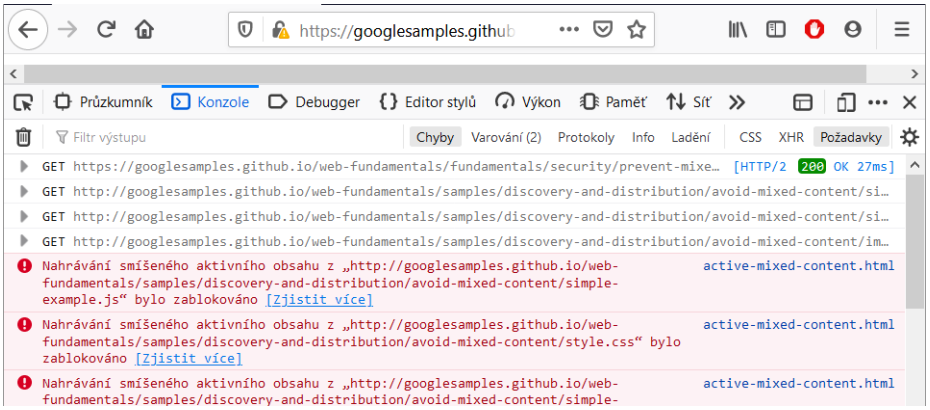
Jednoduché zanesení chyby do konfigurace a související riziko znepřístupnění domény spolu s možností připnutí podvodného klíče k doméně způsobilo, že některé webové prohlížeče odstraňují tuto funkcionalitu. Jedním z dalších důvodů pro její odstranění je i nízké rozšíření této funkcionality v praxi.[7]

1.4.18 Web Packaging

Řada klientů by ocenila možnost přistupovat k webovým stránkám i pokud jsou offline nebo pokud nemají možnost přistoupit ke zdrojovému webovému serveru. Bez připojení ke zdrojovému serveru ale vzniká problém s ověřením pravosti přijatého obsahu. Společnost Google představila technologii Web Packaging. Ta umožňuje svázat zdroje webové stránky a vytvořit z nich jediný soubor. Tento soubor lze kryptograficky podepsat a následně distribuovat klientům dokonce i bez podpory zabezpečeného HTTP spojení. Díky podpisu lze ověřit jejich původ a pravost. Distribuci lze provádět z libovolných webových serverů, cachovacích serverů i dalších zařízení, jako jsou například mobilní telefony.[27]

1.4.19 Hodnota `User-Agent` v hlavičce `CONNECT`

Při vytváření zabezpečeného HTTP spojení s použitím proxy serveru zasílá klient na proxy požadavek `CONNECT`. Tento požadavek má vlastní parametry v hlavičce. Webové prohlížeče přijmou od klienta `CONNECT` požadavek,



Obrázek 1.3: Ukázka problému se smíšeným obsahem

který následně přeposílají na proxy server. Některé webové prohlížeče parametr `User-Agent` pouze přeposílají. Jiné však od této praxe odstupují a parametr nahrazují vlastní defaultní hodnotou.[7]

1.4.20 Smíšený obsah

Po načtení souboru webové stránky přes zabezpečené HTTP spojení jsou klientovi za normálních okolností zaručeny tyto skutečnosti:[28]

- Komunikace klienta se serverem je šifrována, a tak nemůže být jednoduše odposlechnut její obsah zařízeními, přes které procházela. Důvěrnost dat tedy byla zajištěna.
- Klient si mohl ověřit identitu zdrojového serveru. Byla mu tedy umožněna autentizace zdroje.
- Přenášená data nebylo možné jednoduše modifikovat během přenosu, aniž by to klient odhalil. Byla tedy zajištěna integrita dat.

Síla těchto tvrzení může být silně oslabena, pokud jsou některé z dalších součástí webové stránky (skripty, obrázky atd.) načítány přes nezabezpečené HTTP spojení. Přenos takových částí totiž není chráněný před případným útokem typu MITM. Nelze tak zaručit žádné z předchozích tvrzení. Tento problém se u webových stránek vyskytuje poměrně často. Na obrázku 1.3 je zobrazený výpis z konzole prohlížeče Firefox. Ve výpisu je červeně zvýrazněný obsah, který nebyl načtený kvůli přístupu přes nezabezpečené spojení, jelikož prvotní požadavek byl zpracován přes zabezpečené spojení.[28]

Jak vyplývá z obrázku 1.3, některé webové prohlížeče mohou v případě smíšeného obsahu blokovat nezabezpečené zdroje. Některé další webové prohlížeče už ale automaticky upravují url adresu tak, aby se ke zdroji přistupovalo

přes zabezpečené spojení. K tomu stačí nahradit „http://“ na začátku url za „https://“. Zatím je tato funkce použita jen pro vybrané typy objektů, jako je audio a video obsah.[7]

1.4.21 Parametr `SameSite` u cookies

Cookies slouží webovému serveru k uložení dat u klienta. Obvykle se ukládají tokeny nebo stavy webových stránek. Server je může nastavit v hlavičce HTTP odpovědi. Každá položka cookie je pár klíč a hodnota s řadou dalších atributů. Ty určují, kdy a kde se cookie použije, jaká je doba její platnosti, jestli smí být zaslána pouze přes zabezpečené spojení, atd.[29]

Při dotazu na webový server zasílá klient cookies pro různé domény. Podle toho lze cookies rozdělit na ty, které odpovídají doméně aktuální webové stránky (cookies prvních stran), a ty, které této doméně neodpovídají (cookies třetích stran). Parametr `SameSite` umožňuje serveru určit, jak má klient s různými druhy cookies zacházet. Možné hodnoty pro tento parametr jsou:[29]

- `Strict`

  Pro toto nastavení se cookies zašlou pouze pokud jsou v daném kontextu cookies prvních stran. Tedy pokud doména cookies odpovídá aktuální webové stránce (doméně v URL liště prohlížeče). Typ `Strict` se využívá pro cookies související s činnostmi, které uživatel na webové stránce vykonává.

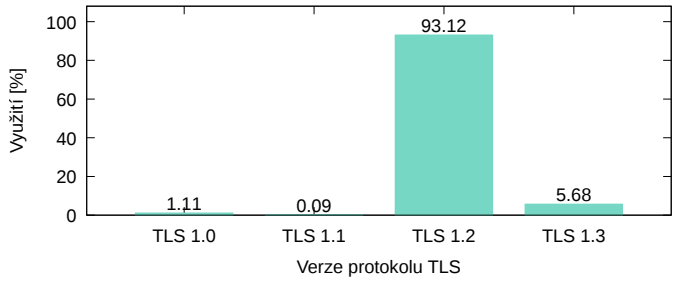
- `Lax`

  Toto nastavení umožňuje odeslat cookies původní webové stránky, pokud na ni uživatel přistupuje pomocí odkazu vyskytujícího se na jiné webové stránce. Cookies se ale nezašlou, pokud je odkaz na původní stránku použit pouze pro stažení určitého obsahu do aktuální webové stránky. Hodnota `Lax` se využívá zejména pro cookies ovlivňující zobrazení webové stránky.

- Atribut není nastaven (ekvivalent hodnoty `None`)

  Tato možnost byla před zavedením atributu SameSite implicitní a znamená, že cookies budou poslány z libovolného kontextu.

Pokud není parametr `SameSite` specifikovaný, jsou cookies náchylné na útoky typu cross-site request forgery (CSRF). Některé webové prohlížeče se proto rozhodly v takovém případě defaultně nastavovat atribut SameSite na hodnotu Lax. Tím dokáží chránit uživatele před některými typy útoků typu CSRF. Zároveň webové prohlížeče mění přístup ke cookies s atributem `SameSite=None`, pokud u nich není specifikován atribut `Secure` (cookies smí být přenášeny pouze přes zabezpečené spojení). Takové cookies jsou webovým prohlížečem odmítnuty. K tomu je přistoupeno, jelikož cookies třetích stran



Obrázek 1.4: TLS verze protokolu a jejich využívání ve webovém prohlížeči Firefox beta 62 (srpen až říjen 2018)

bývají zneužívány útočníky ke sledování chování uživatelů a mohou obsahovat citlivá data týkající se identity uživatele. Díky odmítnutí těchto cookies jsou uživatelé chráněni před útoky na identifikační data uživatelů a zároveň tak lze přimět subjekty k podpoře zabezpečeného HTTP spojení.[7]

1.4.22 TLS 1.0 a TLS 1.1

Protokol TLS 1.0 byl vydaný v roce 1999, tedy již před více než dvaceti lety, TLS 1.1 pak o sedm let později.[30] A ačkoliv nejsou známé konkrétní kritické problémy v těchto protokolech, které by vyžadovaly okamžitou reakci, jsou dnes již používána jiná vhodnější a robustnější řešení. Navíc tyto verze nepodporují dnešní moderní kryptografické algoritmy, ale naopak se spoléhají na algoritmy SHA-1 či MD5. TLS 1.2 řeší slabiny předchozích verzí a navíc podporuje i nové protokoly jako je HTTP/2. Jak je možné vidět na grafu 1.4, TLS 1.2 se společně s novější verzí TLS 1.3 stávají majoritními protokoly pro zabezpečená spojení. Z těchto důvodů webové prohlížeče upozorňují své uživatele na zastaralost protokolů TLS 1.0 a 1.1 a následně přistupují k postupnému ukončování jejich podpory.[31]

1.4.23 Opatření proti TLS downgrade útokům

Během TLS handshake si klient se serverem vyměňují podporované šifrové sady. Následně vybírají šifrovou sadu a parametry TLS spojení, které oba podporují, a které jsou co možná nejodolnější proti útokům. Downgrade útok na TLS se snaží oklamat klienta se serverem, aby použili pro vzájemnou komunikaci starší verze protokolů, nebo parametry TLS spojení, které nejsou příliš bezpečné. Tyto protokoly mohou být podporovány kvůli zpětné kompatibilitě se staršími zařízeními. Díky tomu může útočník využít známé chyby v použitých protokolech, které se vyskytly v TLS komunikaci právě díky Downgrade útoku. K provedení útoku musí útočník zachytit a upravit komunikaci mezi klientem a serverem.[32]

Protokol TLS 1.3 se snaží zabránit tomuto druhu útoku. Podle RFC má být výsledek vyjednávání parametrů stejný v případě Downgrade útoku, jako kdyby k útoku nedošlo. K ověření, že vyjednané parametry nebyly změněny během přenosu, je použit Message Authentication Code (MAC). Ten se použije na konci handshake přes všechny předchozí zprávy. Pokud navíc server zjistí, že jediná použitelná verze protokolu je TLS 1.2 nebo starší, musí ve zprávě `ServerHello` nastavit 8 bytů náhodného čísla na předdefinovanou hodnotu. Pokud klient s verzí TLS 1.3 při kontrole tohoto čísla najde tuto hodnotu, musí spojení ukončit. Pokud by se útočník pokusil změnit hodnotu náhodného čísla, došlo by k chybě při procesu výměny klíčů, během kterého je náhodná hodnota používána.[32]

I když jsou opatření proti Downgrade útoku v TLS 1.3 zpětně kompatibilní, způsobují u některých proxy serverů přerušení spojení. Z tohoto důvodu přistoupily některé webové prohlížeče na dočasné vypnutí nových rozšíření pro kontrolu TLS Downgrade u spojení, která mají řetězec certifikátu s neznámým kořenovým certifikátem. Nyní proklamují zavedení těchto rozšíření do praxe.[7]

1.4.24 Stahování ze zabezpečeného kontextu

Webové prohlížeče zamýšlejí blokovat stahování souborů přes nezabezpečené spojení, pokud bylo iniciováno ze zabezpečeného kontextu. Nejprve budou uživatelé varováni a později se přikročí k blokování. Díky tomuto opatření bude sníženo riziko pro uživatele, protože po stažení může škodlivý soubor obejít ochrany webového prohlížeče.[7]

1.4.25 Zastarání nezabezpečeného HTTP

Nejen společnosti vyvíjející webové prohlížeče se shodují, že v budoucnu bude nutné používat šifrování pro veškerá data přenášená přes Internet. V případě webových stránek to znamená používání zabezpečených HTTP spojení, tedy HTTPS. Některé webové prohlížeče již zveřejnily svůj záměr postupně ukončit podporu nezabezpečeného HTTP spojení. Snaží se tedy definovat, které nové funkcionality již nebudou podporovány pro tato spojení. Zároveň se webové prohlížeče snaží stanovit časový harmonogram postupného omezování dnes podporovaných funkcionalit pro nezabezpečená spojení, které bude nutné v budoucnu odebrat či omezit s ohledem na bezpečnost uživatelů. Zároveň je však připouštěno, že k úplnému odstranění podpory nezabezpečeného HTTP spojení z webových prohlížečů v brzké době nedojde.[33]

1.5 Souhrn

V tabulce 1.1 můžeme porovnáním jednotlivých webových prohlížečů zjistit, že ne všechny změny byly provedeny všemi dnes široce rozšířenými webovými prohlížeči. Příčinou těchto rozdílů může být například odlišná uživatelská

Tabulka 1.1: Srovnání vývoje šifrovaného HTTP v aktuálních webových prohlížečích během posledních tří let

Feature	Google Chrome[1]		Firefox[2]		Safari[3]	
	verze	datum	verze	datum	verze	datum
1.4.1	56	25.1.2017	47	7.7.2016	-	-
1.4.2	56	25.1.2017	-	-	-	-
1.4.3	56	25.1.2017	53	19.4.2017	-	- [4]
1.4.4	56	25.1.2017	43	platné od 1.1.2017	13.0.3	30.10.2019
1.4.5	58	19.4.2017	-	-	-	-
1.4.6	58	19.4.2017	52	7.3.2017	-	-
1.4.7	58	19.4.2017	-	-	13.0.3	30.10.2019
1.4.8	59	5.6.2017	-	pouze varování	-	pouze varování
1.4.9	61	5.9.2017	-	průběžně[34]	12.1.1	24.9.2018
1.4.10	62	17.10.2017	-	-	-	-
1.4.11	65	6.3.2018	61	26.6.2018	12.2	25.3.2019
1.4.12	69	4.9.2018	-	-	-	-
1.4.13	70	16.10.2018	60	9.5.2018	-	-[5]
1.4.14	70	16.10.2018	69[6]	3.9.2019	-	-
1.4.15	70	16.10.2018	-	-	-	-
1.4.16	70	16.10.2018	63	23.10.2018	-	1.8.2018[7]
1.4.17	72	29.1.2019	72	7.1.2020	-	1.8.2018
1.4.18	73	12.3.2019	-	-	-	-
1.4.19	73	12.3.2019	-	-	-	-
1.4.20	80	4.2.2020	-	připravuje se	-	připravuje se
1.4.21	80	4.2.2020	-	připravuje se	-	-
1.4.22	81	13.2.2020	74	10.3.2020	-	květen 2020
1.4.23	81	13.2.2020	72	7.1.2020	12.2	25.3.2019
1.4.24	82	duben 2020	-	-	-	-
1.4.25	-	-	-	30.4.2015	-	-

základna jednotlivých webových prohlížečů s odlišnými preferencemi či rozdílná politika jednotlivých společností. Některé preferují konzervatismus a opatrnou zdrženlivost, jiné se snaží podílet na vývoji nejnovějších technologií a na prosazování těchto technologií do praxe. Zatímco určité webové prohlížeče ruší podporu některých zastarávajících funkcionalit, jiné vyčkávají na vhodnější příležitost, kdy bude zmíněná funkcionalita odstraněna v rámci celého logického celku, či preferují zachování zpětné kompatibility. Případně mohou webové prohlížeče zavádět nové funkce, které po několikaměsíční či roční podpoře opět odstraní, zatímco jiné je na základě svých uvážení nikdy nezačaly podporovat. Další webové prohlížeče se mohou soustředit na snadnou použitelnost, vysoký výkon či na velké množství doplňujících rozšíření. Proto nelze na základě tabulky 1.1 jednoduše hodnotit kvality jednotlivých webových prohlížečů.

[1] Zdroj informací: `www.chromestatus.com/features`

[2] Zdroj informací: `www.mozilla.org/en-US/firefox/releases/`

[3] Zdroj informací: `webkit.org/` a `developer.apple.com/`

[4]Odebráno pouze ECDSA s SHA-512

[5]17.1.2018 začal WebKit (renderovací jádro webového prohlížeče Safari) varovat před zastaralostí AppCache

[6]Odebrání informací o procesoru (32/64 bit)

[7]Částečné omezení důvěry, absolutní je plánováno na později.

Kapitola 2

Dopady pro proxy servery

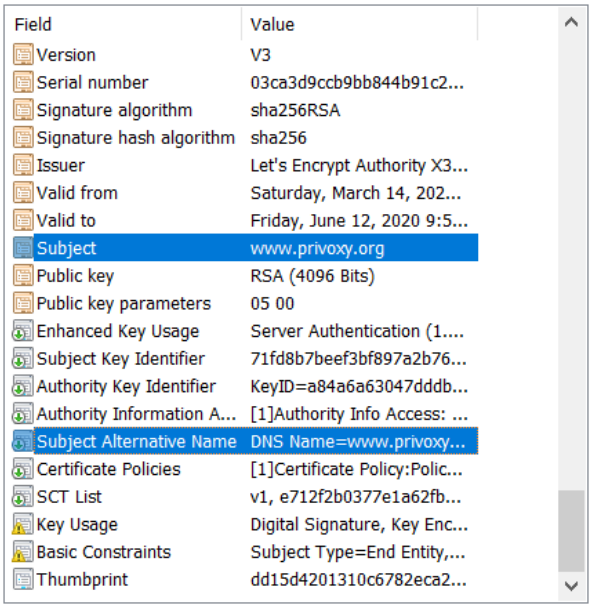
Předchozí kapitola přibližuje vývoj protokolu HTTP a jeho šifrované verze. Zároveň odhaluje i nutnost neustálého vývoje webových prohlížečů, které tyto a mnohé další protokoly využívají. Kromě webových prohlížečů ale existuje i řada dalších prvků, které přenášejí a zpracovávají protokol HTTP a které se rovněž musí neustále přizpůsobovat jeho vývoji. Jedny z těchto prvků jsou proxy servery. Šifrovaná verze protokolu pak sama využívá další prostředky, jako jsou certifikáty a s nimi související infrastruktura veřejných klíčů (PKI). I u webových certifikátů dochází k vývoji a tím i k následnému ovlivnění protokolu HTTPS. V první části tato kapitola popisuje právě změny týkající se webových certifikátů. Druhá část se zaměřuje na proxy servery a na změny, které je nezbytné či vhodné implementovat, aby mohly i nadále filtrovat přenášenou HTTPS komunikaci.

2.1 Změny v certifikátech

Při navazovaní HTTPS spojení mezi klientem a webovým serverem může dojít v průběhu TLS handshake k výměně certifikátů. Certifikát může zaslat server i klient, pro protokol TLS se obvykle jedná o typ X.509v3. Pokud byl certifikát zaslán, pak slouží k autentizaci protější strany. Aby nebyl certifikát odmítnut jako nevalidní, musí splňovat všechny náležitosti včetně nejnovějších bezpečnostních požadavků.

2.1.1 Parametr `Subject Alternative Name`

Jak již bylo zmíněno, při kontrole certifikátu webového serveru je nutné porovnat hostname serveru s hostname uvedeným v certifikátu, který obdržel. Na obrázku 2.1 je zobrazen seznam parametrů, které obsahuje certifikát serveru `www.privoxy.org`. Certifikát obsahuje parametr `Subject Common Name` i `Subject Alternative Name` s hodnotou `DNS Name`. Podle RFC 2818 by tedy



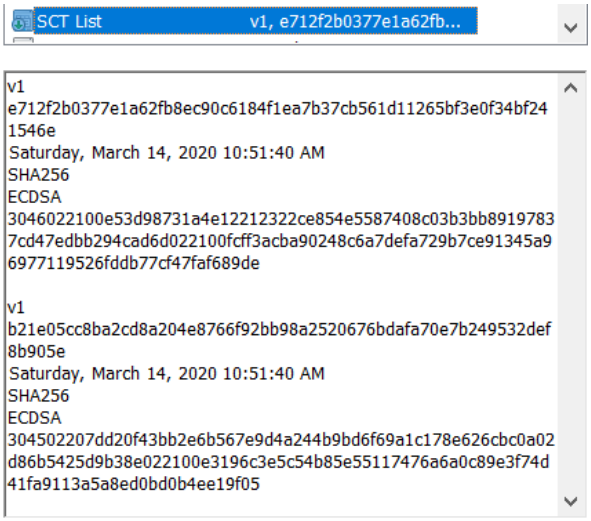
Obrázek 2.1: Parametry certifikátu `www.privoxy.org`

měla být pro kontrolu hostname tohoto certifikátu použita hodnota `Subject Alternative Name`.

Aby mohl proxy server filtrovat přenášenou HTTPS komunikaci, musí použít metodu MITM. Při použití této metody navazuje proxy server TLS spojení s klientem, a aby autentizace webového serveru z pohledu prohlížeče proběhla úspěšně, musí proxy server zaslat klientovi platný certifikát s odpovídajícím hostname. Pokud je v certifikátu nastaven pouze parametr `Subject Common Name`, starší verze webových prohlížečů použijí tento parametr. Novější verze však vyžadují parametr `Subject Alternative Name` a pokud ho certifikát neobsahuje, označí ho jako nevalidní a varují uživatele před přístupem na danou webovou stránku. Z tohoto důvodu musí začít proxy servery do generovaných certifikátů tento parametr vkládat. Některé kryptografické knihovny ovšem neumožňují tento parametr nastavit, a tak musí proxy server zvolit vhodnou knihovnu.

2.1.2 Certificate Transparency

Certificate Transparency umožňuje webovým prohlížečům ověřit, že přijatý certifikát nebyl vydaný certifikační autoritou chybně. K tomuto ověření využijí časové razítko SCT podepsané logem, do kterého byl certifikát vložen. Jednou z možností, jak může webový prohlížeč SCT získat, je certifikát serveru, který jej zašle během TLS handshake. Do certifikátu je SCT ukládáno CA během jeho vystavování. CA nejprve vytvoří tzv. předcertifikát, který zašle



Obrázek 2.2: Ukázka parametru SCT list v certifikátu `www.privoxy.org`

do CT logu a obratem od něj obdrží SCT.[14] Předcertifikát obsahuje totožná data jako klasický certifikát, ale jsou navíc doplněna rozšířením (tzv. poison extension) s předdefinovanou hodnotou. Tato hodnota slouží k ujištění, že předcertifikát nebude použit k navázání TLS spojení ani k ověření identity webového serveru. SCT získané od CT logu pak CA přidá do nově vytvářeného certifikátu v rozšíření SCT list. Je vhodné, aby certifikát obsahoval SCT z několika různých CT logů, stejně jako v certifikátu na obrázku 2.2.[15]

2.1.3 Podporované algoritmy

Společně s protokoly HTTP a TLS dochází k vývoji i u algoritmů, které jsou těmito protokoly využívány. Zároveň vznikají nové techniky k prolomení těchto algoritmů a roste i výkon zařízení, která mohou útočníci k prolomení algoritmů využít.[9] Z těchto důvodů webové prohlížeče odstraňují některé algoritmy ze seznamu podporovaných algoritmů pro zabezpečenou komunikaci. Jako příklad můžeme uvést algoritmus SHA-1, který byl označen za zastaralý a který mohou útočníci zneužít k padělání certifikátu. Webové prohlížeče následně přestaly akceptovat certifikáty, které obsahují podepsaný hash vytvořený pomocí SHA-1. Webové servery tak musí nahradit staré certifikáty využívající SHA-1 novými, které mají podepsaný hash vytvořený např. algoritmem SHA-256. Kromě hashovacích algoritmů přestávají webové prohlížeče akceptovat i zastaralé algoritmy sloužící k digitálním podpisům, případně určité kombinace podpisového algoritmu a hashovací funkce. Jednou z takových kombinací je i ECDSA s SHA-1 a s SHA-512.[7] Rovněž tyto změny musí zohlednit webové servery ve svých certifikátech.

2.1.4 Zneplatnění certifikátů

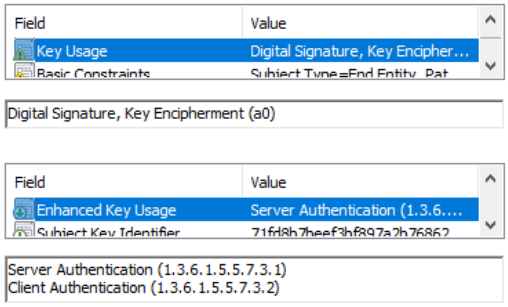
S opakujícími se problémy CA při vydávání certifikátů přicházejí i nové způsoby, jak se s nimi vyrovnat. Certificate Revocation List (CRL) je seznam certifikátů, které byly vydávající CA zneplatněny před datem ukončení platnosti uvedeným v certifikátu. Během ověřování certifikátu může webový prohlížeč nebo proxy server ověřit, že certifikát není uveden v tomto seznamu. CRL ovšem spravuje CA, která certifikát vydala, a tudíž je pro zneplatnění certifikátu nutná její spolupráce. Pokud CA vydala určitý certifikát chybně a nespolupracuje při jeho revokaci, je potřeba využít jiné mechanismy.

Proxy server či webový prohlížeč mohou využít služeb CT. Pro CT není nutná spolupráce CA, jelikož certifikát do CT logu může nahrát kdokoli. Čím více organizací se ovšem zapojí do nahrávání záznamů do CT logů, tím vyšší zabezpečení může služba CT poskytnout.[14] Po odhalení problémů s certifikáty vydanými společností Symantec využil Google právě CT. A tak začal webový prohlížeč Chrome vyžadovat zapsání všech nově vydaných certifikátů v CT logu nejprve u certifikátů vydaných právě společností Symantec.[35] Pokud certifikát této společnosti nebyl zapsaný v CT logu, varoval uživatele před neplatným certifikátem a zastavil načítání webové stránky. Tím byl umožněn lepší dohled nad těmito certifikáty.

Webový prohlížeč Chrome nakonec zcela ukončil důvěru pro všechny certifikáty vydané společností Symantec pomocí původní PKI.[25] To způsobilo, že ačkoliv byly certifikáty po formální stránce platné, prohlížeč Chrome je i tak označil za neplatné a varoval uživatele. Jiné webové prohlížeče ale totožné certifikáty přijaly jako validní a uživatelům umožnily přístup na dotazovanou webovou stránku. V případě použití proxy serveru, který filtruje HTTPS komunikaci, obdrží webový prohlížeč certifikát vygenerovaný proxy serverem. Záleží tedy pouze na proxy serveru, jestli označí skutečný certifikát serveru za validní a umožní klientovi načíst dotazovanou webovou stránku, nebo jestli spojení ukončí a klienta varuje. V takovém případě by tedy byla webová stránka využívající certifikát společnosti Symantec zpřístupněna i v prohlížeči Chrome bez varování. Aby bylo takové spojení ukončeno i s využitím proxy serveru, bylo by nutné rozšířit konfiguraci o blokování libovolného certifikátu, případně o blokování na základě určitých parametrů.

2.1.5 Účel certifikátu

Během ověřování platnosti certifikátu může být zohledněn i účel, pro který byl certifikát vydán, respektive ke kterým účelům lze použít veřejný klíč certifikátu. Účely certifikátu mohou být definovány pomocí dvou parametrů: `Key Usage` a `Extended/Enhanced Key Usages`. Některé z možných účelů jsou: autentizace serveru, autentizace klienta, šifrování, dešifrování, podpis certifikátů, ... Pokud tedy webový prohlížeč ověřuje identitu severu pomocí certifikátu, musí k tomu být přijatý certifikát určený. Při použití proxy serveru



Obrázek 2.3: Nastavení účelu pro certifikát `www.privoxy.org`

k dešifrování HTTPS komunikace tedy musí proxy sever v certifikátu tyto parametry korektně nastavit, a to pro všechny certifikýty v řetězci včetně certifikátu CA. Na obrázku 2.3 je znázorněno nastavení těchto parametrů pro `www.privoxy.org`.

2.2 Změny pro proxy servery

Proxy servery filtrující HTTPS komunikaci musí reagovat na vývoj tohoto protokolu i na změny ve webových prohlížečích. Změny se mohou týkat parametrů šifrovaného spojení, podporovaných protokolů, ale i nových funkcionalit. Některé změny v proxy serveru jsou nezbytné pro správnou funkčnost, jiné pouze rozšiřují již podporované funkcionality proxy serveru.

2.2.1 HTTP/2

Protokol HTTP/2 vyžaduje značná rozšíření proxy serverů podporující pouze starší verze protokolu HTTP. Od předchozích verzí se totiž velmi zásadně liší, jak je popsáno v předchozí kapitole. Následující seznam uvádí nejzásadnější změny, které musí proxy server implementovat pro podporu HTTP/2:[3]

- Jelikož je protokol HTTP/2 na rozdíl od předchozích verzí binární, je nutné implementovat zcela nový způsob parsování přijatých rámců. Zároveň je nezbytné filtrovaná data před odesláním opět převést do binárního formátu.
- Komprese hlaviček je v HTTP/2 implementována pomocí tabulky hlaviček na straně klienta i serveru. Hlavičku pak lze nahradit odkazem od této tabulky. Celý tento mechanismus musí proxy server implementovat pro komunikaci se serverem i s klientem.
- HTTP/2 umožňuje pomocí jediného TCP spojení zpracovávat paralelně řadu požadavků. K tomu slouží tzv. streamy. Každý dotaz má vlastní

stream a každý přenášený paket obsahuje identifikátor tohoto streamu. Klient tak musí dokázat paralelně zpracovat několik dotazů. Díky funkci Server push navíc musí klient zpracovávat i odpovědi, které od serveru nevyžádal.

- Protokol HTTP/2 využívá z důvodu zpětné kompatibility s předchozími verzemi stejné porty, tedy 80 a 443. Jelikož se ale značně liší od předchozích verzí, musí se během navazování spojení na případném použití HTTP/2 domluvit. K tomu slouží rozšíření Application-Layer Protocol Negotiation (ALPN) protokolu TLS. To umožňuje během TLS handshake vyjednat aplikační protokol následné komunikace. Díky tomu může být použita řada různých aplikačních protokolů přes TLS spojení na jednom portu. Proxy servery tak musí využívat knihovnu pro protokol TLS, která toto rozšíření podporuje.

  V případě použití HTTP/2 bez TLS není rozšíření ALPN dostupné. Místo něj odešle klient dotaz pomocí HTTP/1.1 s hlavičkou `UPGRADE`, ve které navrhne přechod na HTTP/2. Pokud na tento požadavek server přistoupí, odešle odpověď se stavovým kódem 101 (Switching protocol) a následně se využívá HTTP/2.

I když proxy server nepodporuje HTTP/2, může klientovi umožnit přenos těchto požadavků pomocí tunelu skrz.[1] Tento postup lze využít i v aktuální verzi proxy serveru Privoxy.[36] Případně může při přijetí požadavku na navázání spojení pro protokol HTTP/2 vyjednat s webovým prohlížečem degradování verze protokolu.

2.2.2 Nové kryptografické algoritmy

Webové prohlížeče implementují nejnovější verze šifrovacích algoritmů a zároveň ukončují podporu pro ty, které jsou již označeny za zastaralé. Z tohoto důvodu by měly i proxy servery používat kryptografické knihovny implementující nejnovější šifrovací algoritmy. Zároveň je vhodné, aby proxy servery umožnily uživatelům definovat specifickou šifrovou sadu podle cílového serveru. Díky tomu lze minimalizovat zranitelnosti jednotlivých spojení. Případně je vhodné umožnit alespoň všeobecné omezení šifrové sady pro všechna TLS spojení globálně. Tato možnost ale neumožňuje udělit výjimku z důvodu zpětné kompatibility.

2.2.3 Certificate Transparency

Certificate Transparency vyžaduje během ověřování platnosti certifikátu, aby byl certifikát zapsaný alespoň v jednom z veřejných CT logů. Avšak CT je zpravidla vyžadováno pouze pro veřejně důvěryhodné certifikáty. To jsou certifikáty podepsané kořenovou CA, která je dodána v instalaci webového

prohlížeče či operačního systému.[37] Pokud přijme prohlížeč od webového serveru nebo proxy serveru certifikát podepsaný soukromou CA, pak k ověřování tohoto certifikátu pomocí CT nedojde. Pokud tedy proxy server filtrující HTTPS komunikaci podepisuje generované certifikáty pro jednotlivá doménová jména pomocí soukromé CA, nemusí tyto certifikáty vkládat do CT logů. A jelikož CT logy dovolují vkládat pouze certifikáty podepsané známou CA, nebylo by to proxy serveru ani umožněno.

I když proxy server nemusí věnovat pozornost CT na straně webového prohlížeče, může provádět kontrolu CT u certifikátů přijatých během autentizace webového serveru. K tomu ale musí využívat kryptografickou knihovnu, která tuto funkci podporuje.

2.2.4 TLS 1.3

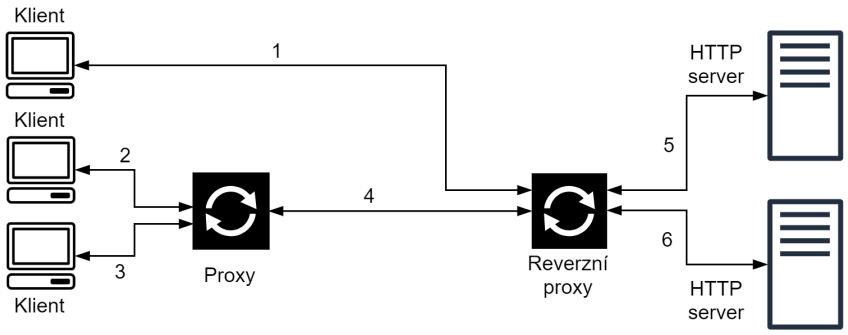
Verze protokolu TLS 1.3 přinesla řadu změn oproti předchozím verzím. Z hlediska proxy serveru je důležité zejména používání kryptografické knihovny, která rychle implementuje tuto novou verzi protokolu a umožní tak proxy serveru její využívání. Veškeré změny přicházející s novou verzí protokolu pak řeší tato knihovna a další přímé důsledky pro proxy server z nich nevyplývají.

2.2.5 Komprese certifikátů

Dokud neumožňoval protokol TLS šifrovat certifikáty přenášené v průběhu TLS handshake, nebyla komprese certifikátů používána právě z důvodu mezilehlých zařízení jako jsou proxy servery. Proxy servery totiž mohou využívat toho, že přenášené certifikáty nejsou během handshake širovány. A i když proxy server nepodporuje filtrování HTTPS komunikace a místo využití metody MITM přenáší tuto komunikaci TLS tunelem skrz proxy, dokáže z přenášených dat certifikát vyexportovat. Pokud takto získaný certifikát označí jako nevalidní, může TLS tunel ukončit. TLS 1.3 však přineslo možnost šifrování a komprese přenášeného certifikátu. Proxy servery nabízející tuto funkcionalitu ji tak musí při použití TLS 1.3 vypnout. Pokud navíc nedokáží odhalit použitou verzi TLS protokolu, musí tuto funkcionalitu zcela odstranit, případně pro její zachování implementovat metodu MITM.

2.2.6 Úprava hodnot `User-Agent`

Změna hodnoty `User-Agent` webovým prohlížečem v hlavičce požadavku je z hlediska proxy serveru redundantní. Pokud proxy server umožňuje filtrovat přenášenou komunikaci včetně HTTP hlaviček, může hodnotu nejen tohoto parametru libovolně měnit podle konfigurace uživatele. Uživatel tak sám může zvolit vlastní strategii, jak předejít jeho identifikaci pomocí nastavených hodnot v HTTP hlavičkách.



Obrázek 2.4: Rozdělení TLS spojení při použití proxy na straně klienta a reverzní proxy na straně serveru

2.2.7 Token Binding

Ačkoliv webové prohlížeče postupně ukončují podporu Token Binding, případně Token Binding nikdy ani nezavedly, proxy servery by měly být připravené i na jeho použití. Na obrázku 2.4 je vidět, na kolik samostatných TLS spojení je rozdělená komunikace mezi klientem a serverem, pokud je na straně klienta použit proxy server filtrující přenášenou HTTPS komunikaci pomocí metody MITM a reverzní proxy server přerušující TLS spojení (TLS terminating reverse proxy TTRP) na straně serveru. Každá očíslovaná šipka na obrázku znázorňuje jedno TLS spojení.

Při použití Token Binding s protokolem HTTPS je token vázán ke konkrétnímu TLS spojení. HTTP server generující token v rámci cookies však při použití TTRP serveru získá informace jen z TLS spojení s ním, například z TLS spojení č. 6. Toto spojení však nemusí být specifické pro konkrétního klienta, a navíc specifikuje jen část spojení mezi klientem a serverem, a tak by svázání tokenu s ním nemělo smysl. Pokud by byl ukradený token poslán útočníkem, přišel by na server opět přes TLS spojení č. 6 a kontrola Token Binding by proběhla úspěšně.[38]

Tento problém je možné vyřešit přidáním nových parametrů do HTTP hlavičky požadavku po jeho přijetí reverzním proxy serverem. TTRP server provede kontrolu Token Binding Message přijaté od klienta, následně z ní vyjme jednotlivé Token Binding ID a přidá je do HTTP hlavičky jako následující parametry. Tyto parametry nesmí být nastavené proxy serverem na straně klienta ani jinými mezilehlými zařízeními.[38]

- `Sec-Provided-Token-Binding-ID`

  Jedná se o typ Token Binding ID, který slouží pro vytvoření Token Binding mezi klientem a serverem, respektive mezi proxy serverem na straně klienta a TTRP serverem.

- `Sec-Referred-Token-Binding-ID`

  Typ Token Binding ID který se používá při žádosti o tokeny, které mají být předloženy na jiný server, například autorizační.

- `Sec-Other-Token-Binding-ID`

  Tento parametr obsahuje seznam dalších typů Token Binding zaslaných klientem a ověřených TTRP serverem.

Na straně klienta musí být Token Binding Message uvedena v prvním HTTP požadavku po navázání TLS spojení s webovým serverem. Pokud ovšem klient využívá proxy sever filtrující HTTPS komunikaci, je token svázán s TLS spojením nejblíže webovému serveru, respektive jeho TTRP. Na obrázku 2.4 je toto spojení označeném číslem 4. K tomuto spojení ovšem klient nemá přístup, a tak pro něj nemůže vytvořit Token Binding Message. Parametry TLS spojení vlastní proxy server, a tak musí na jejich základě vytvořit Token Binding Message a přidat ji do HTTP hlavičky požadavku pomocí parametru `Sec-Token-Binding`. Webový server ji sváže s vytvářenými cookies, které odešle v odpovědi. Proxy server již může cookies přeposlat klientovi, aniž by klient věděl o používání protokolu Token Binding. Pokud klient uvede cookies v dalších požadavcích, klient je opět přepošle webovému serveru a ten ověří, že byly přijaty přes odpovídající TLS spojení.

2.2.8 HTTP Public Key Pinning

HPKP připíná k doménovému jménu konkrétní veřejné klíče, které by měl certifikát přijatý z konkrétní domény obsahovat. Konkrétní klíče jsou přenášeny webovému prohlížeči při prvním načtení webové stránky v parametru HTTP odpovědi. Tyto parametry může proxy server vyfiltrovat, a tím znemožnit klientovi HPKP použít. Pokud se ale webovému prohlížeči podaří přijmout HPKP hlavičku od serveru například v jiné síti, může při příštím připojení přes proxy server konkrétní klíč vyžadovat, a pokud v certifikátu nebude obsažen, spojení i ukončit. Webové prohlížeče jako Chrome a Firefox ale vypínají kontrolu HPKP, pokud je řetězec přijatého certifikátu zakončen kořenovým certifikátem, který uživatel importoval do seznamu důvěryhodných CA. Tedy pokud byl přijatý certifikát podepsaný proxy serverem pomocí CA, která je importovaná v prohlížeči, pak nebude kontrola HPKP provedena, respektive její výsledek bude ignorován.[39] Důsledek tohoto přístupu však umožňuje útočníkům obejít ochranu pomocí HPKP, pokud dokáží vložit vlastní CA do seznamu důvěryhodných CA například pomocí malware. V takovém případě HPKP klienta neochrání.[40] I tento fakt je argumentem pro využití jiných metod ochrany klienta namísto HPKP.

2.2.9 `SameSite` parametr u cookies

Proxy server filtrující HTTPS komunikaci může filtrovat i parametry HTTP hlaviček. Díky tomu může správce proxy serveru libovolně upravovat parametr `SameSite` u cookies v závislosti na doméně, ze které přichází. Například může specifikovat výchozí hodnotu parametru `SameSite`, případně doplnit cookies s `SameSite=None` o atribut `Secure`, aby nebyly webovým prohlížečem odmítnuty. Díky těmto možnostem mohou proxy servery obejít jakékoli nastavení tohoto parametru, případně jej zcela odstranit.

2.2.10 TLS 1.0 a TLS 1.1

Vývoj zabezpečeného HTTP spojení se netýká pouze kryptografických algoritmů, ale i kryptografických protokolů, které tyto algoritmy využívají. Webové servery pak postupně přecházejí na tyto novější verze protokolů, a tím se zvyšuje jejich procentuální zastoupení v rámci HTTPS. Na tuto skutečnost musí reagovat i proxy servery filtrující HTTPS komunikaci. Je vhodné, aby proxy servery umožnily omezit použití zastaralých verzí TLS 1.0 a TLS 1.1 a nahradily je nejnovějšími verzemi. Tyto nové protokoly tak musí být podporovány používanou kryptografickou knihovnou.

2.2.11 Zastarání nezabezpečeného HTTP

Ačkoliv dosud webové prohlížeče zpravidla nestanovily přesné datum pro ukončení podpory nezabezpečeného HTTP protokolu, proklamují, že k této skutečnosti dojde. Proxy servery by se proto měly připravit na tuto skutečnost. Proxy servery jako je Privoxy, které umožňují filtrovat pouze nešifrovaný HTTP protokol, by v takovém případě nemohly uplatnit naprostou většinu svých filtrovacích algoritmů.

Kapitola 3

Návrh úprav

Pro rozšíření proxy serveru Privoxy o možnost filtrovat veškerou komunikaci přenášenou pomocí šifrovaného HTTP spojení a další nové funkcionality je nutné navrhnout a implementovat řadu dílčích změn. Před tím je nezbytné analyzovat výchozí stav zdrojového kódu proxy serveru, ze kterého budou následné úpravy vycházet.

3.1 Proxy server Privoxy

Privoxy je webový proxy server bez mezipaměti, který nabízí řadu funkcí pro filtrování přenášené HTTP komunikace, ale pouze pokud není přenášena pomocí zabezpečeného spojení. Tyto funkce umožňují filtrovat data webových stránek včetně HTTP hlaviček. Filtry mohou sloužit například pro odstranění reklam z webových stránek, či pro zvýšení bezpečnosti díky odstranění potencionálně škodlivých skriptů. K dispozici jsou předdefinované filtry a další akce nad přenášenými daty. Uživatel má zároveň možnost specifikovat filtry podle individuálních potřeb. Během posledních tří let byly vydány dvě nové verze proxy serveru. Nyní tak je nejaktuálnější verzí Privoxy 3.0.28.

3.2 Výstup bakalářské práce

Tato diplomová práce navazuje na výstup bakalářské práce „Podpora pro filtrování SSL/TLS komunikace v Privoxy“.[1] Bakalářská práce rozšiřuje proxy sever Privoxy ve verzi 3.0.26 o možnost částečně filtrovat přenášenou HTTPS komunikaci. Níže jsou popsány jednotlivé implementované části.

3.2.1 Metoda Man In The Middle

Pro úspěšné filtrování HTTPS komunikace bylo nutné implementovat princip MITM, tedy vytvoření dvou samostatných TLS spojení, jedno mezi klientem a proxy serverem a druhé mezi proxy serverem a webovým serverem.

Tento postup nahradil pro HTTPS komunikaci původní přenos dat pomocí TLS tunelu, tedy pouhé přeposílání šifrovaných dat mezi klientem a serverem přes TCP spojení. Díky principu MITM jsou přenášená data dešifrována a mohou na ně být následně aplikovány již existující filtrovací algoritmy implementované v Privoxy. Následně jsou data opět zašifrována a zaslána na cílové zařízení. Tento princip umožňuje filtrovat komunikaci v obou směrech, tedy jak data přijatá od klienta, tak data od serveru, a to včetně HTTP hlaviček. Zároveň bakalářská práce umožňuje filtrování HTTPS komunikace i při použití nadřazeného proxy serveru, i když se v takovém případě navazování spojení s cílovým webovým serverem liší. Pro TLS komunikaci byla v bakalářské práci použita kryptografická knihovna Mbed TLS.

3.2.2 Částečné filtrování komunikace

Původní verze proxy serveru Privoxy již nabízí sadu funkcí pro filtrování textových dat přenášených pomocí nezabezpečeného HTTP spojení, včetně funkcí provádějících dekompresi a kompresi dat. K manipulaci s daty je možné použít filtry definované uživatelem v konfiguračním souboru, případně předdefinované filtry, a nebo sofistikovanější akce, které jsou implementovány přímo ve zdrojovém kódu. Tyto filtry se následně přiřazují ke konkrétním url adresám. Na tento základ navázala bakalářská práce, která umožnila aplikovat zmíněné filtry a akce na soubory i HTTP hlavičky přijaté přes zabezpečené HTTP spojení, ale pouze pokud byly přijaty od cílového webového serveru. Filtrování dat od klienta nebylo kvůli nezbytným úpravám velkého rozsahu v rámci bakalářské práce realizováno.

3.2.3 Dynamické vytváření certifikátů

Při využití metody MITM navazuje proxy server dvě samostatná TLS spojení. Nejdříve přijme od klienta HTTP požadavek na navázání spojení s požadovaným webovým serverem. Tento požadavek je přijat přes nezabezpečené spojení. Proxy server následně vytváří TLS spojení s webovým serverem a poté i s klientem. Webový prohlížeč ověřuje identitu protější strany pomocí certifikátu přijatého v průběhu TLS handshake. Pokud by prohlížeč označil certifikát jako neplatný, upozornil by uživatele a spojení ukončil. Aby k tomu nedocházelo, musí proxy server zaslat validní certifikát, který bude prohlížečem uznán jako platný pro požadovanou url adresu.

Z těchto důvodů bylo v rámci bakalářské práce implementováno i dynamické vytváření certifikátů a k nim příslušejících párů soukromého a veřejného klíče. Ty jsou vytvářeny pro každou doménu, na kterou klient zašle požadavek. Proxy server certifikáty podepisuje vlastní CA, kterou je zároveň nezbytné importovat do webového prohlížeče klienta. Pro vytváření certifikátů byla v bakalářské práci použita knihovna Mbed TLS, která však neumožňuje nastavit některá rozšíření vytvářeného certifikátu. Jedním z těchto rozšíření

je i `Subject Alternative Name`. Jelikož začaly některé webové prohlížeče striktně vyžadovat rozšíření typu `Subject Alternative Name` pro ověření identity webového serveru, začaly zároveň kvůli absenci tohoto parametru v přijatých certifikátech označovat veškerá TLS spojení s proxy serverem za nedůvěryhodná.

3.2.4 Kontrola certifikátů na straně proxy serveru

Jak již bylo uvedeno, při filtrování HTTPS komunikace pomocí proxy serveru je vytvářeno TLS spojení mezi webovým prohlížečem a proxy serverem. Jelikož proxy server zasílá webovému prohlížeči certifikát, který sám vytvořil, nezíská prohlížeč původní certifikát webového serveru, a tak nemůže ani ověřit platnost tohoto certifikátu. Z tohoto důvodu musí kontrolu certifikátu webového serveru provádět již proxy server, který s tímto serverem navazuje TLS spojení. Tato kontrola certifikátů byla implementována již v rámci bakalářské práce a k jejímu provedení využívá proxy server seznam důvěryhodných CA vytvořený správcem serveru. Pokud by proxy server tuto kontrolu neprováděl a všechny certifikáty automaticky označil jako validní, navázal by TLS spojení s klientem a zaslal mu jím vytvořený certifikát. Ten by prohlížeč přijal jako validní a zahájil by s webovým serverem komunikaci. Díky tomu by bylo HTTPS spojení navázáno s jakýmkoli webovým serverem bez ohledu na jeho certifikát. Uživatel by tak zůstal před případnými útoky na jeho komunikaci s webovým serverem naprosto nechráněný.

Pokud proxy server označí certifikát jako neplatný, přeruší navazování TLS spojení s webovým serverem, ale přesto naváže TLS spojení s klientem s využitím platného certifikátu. Následně však tímto spojením zašle klientovi HTML stránku obsahující informace o chybě během kontroly certifikátu, včetně základních informací o všech certifikátech v podpisovém řetězci. Zároveň bakalářská práce umožňuje uživateli tyto certifikáty stáhnout pomocí odkazu. Díky tomu má uživatel detailní informace o důvodech, pro které byl certifikát odmítnut.

3.2.5 Konfigurace

Součástí bakalářské práce je i rozšíření možností konfigurace proxy serveru o konfiguraci nově implementovaných funkcionalit. Nastavit lze jednak parametry pro TLS komunikaci, jako je například CA pro podpis nově vytvářených certifikátů, nebo soubor obsahující důvěryhodné CA použité při kontrole certifikátů webových serverů. Dále bylo Privoxy v rámci bakalářské práce rozšířeno o dvě nové akce, které může správce proxy serveru přiřadit jednotlivým url adresám. Pomocí akce definované klíčovým slovem `disable-ssl-filtering` lze pro protokol HTTPS využít místo metody MITM původní TLS tunel. Druhé klíčové slovo `ignore-certificate-errors` umožňuje vypnout kontrolu certifikátů pro zvolené webové servery.

3.3 Nově implementovaná rozšíření

V rámci této diplomové práce je implementována řada nových funkcionalit a vylepšení navazujících na předchozí bakalářskou práci. Zároveň je provedena řada změn ve struktuře a zpracování zdrojového kódu. Hlavní cíle práce jsou uvedeny v této části.

3.3.1 Zpracování výstupů bakalářské práce

Výstup bakalářské práce není nijak strukturovaný, ani není vložený do žádného verzovacího systému. To komplikuje jeho následné zpracování třetími stranami a případné nasazení do produkčního prostředí. Z tohoto důvodu je vhodné provést před samotnou implementací nových rozšíření revizi výchozího zdrojového kódu a následně rozdělení úprav implementovaných v bakalářské práci do menších logických celků. Současně je vhodné tyto celky vložit do nově vytvořeného repozitáře umožňujícího správu verzí.

Během posledních tří let došlo k vydání dvou nových verzí proxy serveru Privoxy. Nejaktuálnější je nyní verze 3.0.28. Původní bakalářská práce ovšem rozšiřuje verzi 3.0.26. Před zahájením implementace dalších navazujících rozšíření je tedy vhodné spojit zdrojový kód bakalářské práce s nejnovější verzí Privoxy. Díky předchozímu rozdělení bakalářské práce na menší logické celky je tato operace jednodušší. Díky využití verzovacího systému i pro implementaci diplomové práce bude zjednodušeno i její případné připojení k dalším vydaným verzím Privoxy.

3.3.2 Přizpůsobení webovým prohlížečům

Webové prohlížeče neustále vydávají nové verze, ve kterých implementují nové funkcionality i nová bezpečnostní opatření. Je nezbytné, aby se proxy server těmto změnám přizpůsobil. Jedině tak může být zaručeno, že při spolupráci s webovými prohlížeči nebude docházet k neočekávaným událostem, jako je například náhlé ukončení spojení.

Jednou z takových důležitých změn, která byla vydána během posledních tří let, je kontrola doménového jména v certifikátech pouze podle hodnoty rozšíření `Subject Alternative Name`. Proxy server tak musí v certifikátech toto rozšíření korektně nastavit a k tomu je nezbytná vhodná kryptografická knihovna, která umožňuje tento parametr ve vytvářených certifikátech nastavit. Jelikož kryptografická knihovna Mbed TLS zvolená v rámci bakalářské práce neumožňuje nastavení hodnoty pro toto rozšíření, je nutné zvolit novou, která bude lépe vyhovovat účelům proxy serveru. Zároveň je nezbytné implementovat veškeré funkcionality z bakalářské práce i s využitím této nové knihovny tak, aby bylo co nejvíce zachováno rozhraní původních funkcí. Díky tomu bude možné zkompilovat proxy server s původní nebo novou knihovnou podle preferencí uživatele.

3.3.3 Umožnění kompilace bez kryptografických knihoven

Díky zachování stejného rozhraní u funkcí využívajících novou kryptografickou knihovnu, jako mají původní funkce implementované v bakalářské práci, bude možné pomocí maker v kódu umožnit kompilaci s novou či původní knihovnou. Zároveň lze díky makrům upravit zdrojové kódy tak, aby se uživatel mohl rozhodnout zkompilovat proxy server v původní podobně, tedy bez kryptografické knihovny a bez rozšíření umožňujícího filtrovat HTTPS komunikaci. Rozšíření vstupních souborů do konfiguračních skriptů pak umožní uživateli pomocí přepínačů zvolit preferovaný způsob kompilace bez jakýchkoli změn v kódu. Konfigurační skripty následně vygenerují odpovídající `makefile`.

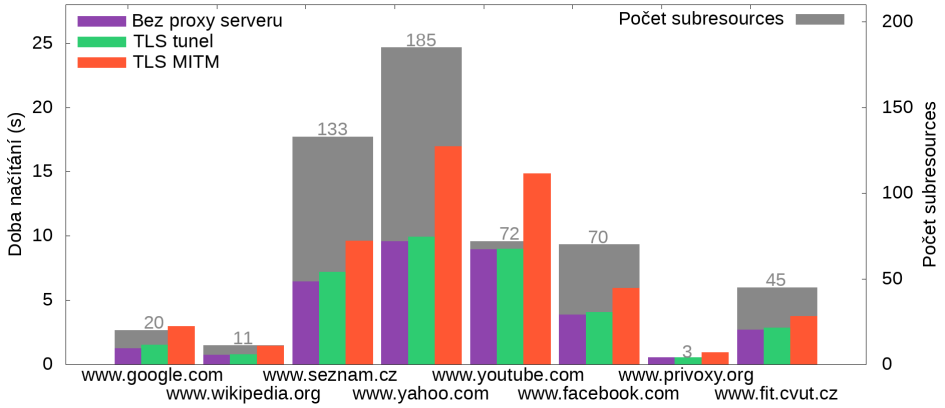
3.3.4 Filtrování veškeré přenášené HTTPS komunikace

Hlavním cílem této diplomové práce je modernizovat proxy server Privoxy tak, aby bylo možné jeho veškeré funkcionality pro filtrování HTTP komunikace využít i pro komunikaci přenášenou přes zabezpečené spojení, tedy pomocí protokolu HTTPS. Tato práce navazuje na bakalářskou práci, která již umožňuje použít tyto funkcionality pro data přijatá od webového serveru. Je tedy potřeba upravit načítání požadavků webového prohlížeče tak, aby i v případě využití HTTPS komunikace byly tyto požadavky naparsovány. Na základě takto získaných dat pak bude možné aplikovat uživatelem definované akce upravující parametry HTTP dotazu. Poté se dotaz opět sestaví z upravených parametrů a odešle pomocí šifrovaného spojení na webový server.

3.3.5 Konfigurační rozhraní přes HTTPS

Proxy server Privoxy umožňuje zobrazit a editovat konfiguraci některých parametrů pomocí webové stránky dostupné na specifické url adrese. Zároveň je možné na této url ověřit aplikované filtry pro zadanou url, případně ověřit výstup po aplikování určitých filtrů. Toto konfigurační rozhraní je ovšem v proxy serveru Privoxy dostupné pouze přes nezabezpečené spojení. Pokud uživatel zadá dotaz na konfigurační url s předponou „https://“, pak Privoxy nevrátí konfigurační rozhraní, ale pokusí se dotaz přesměrovat na konkrétní webový server, který zpravidla není dostupný.

Proxy server nemusí být vždy spuštěný přímo na lokálním zařízení uživatele, ale například na serveru, routeru nebo jiném síťovém zařízení. Je tedy vhodné, aby případná komunikace mezi těmito dvěma zařízeními, která může upravovat konfiguraci proxy serveru, byla zabezpečená. To umožní zabránit odposlechu komunikace v rámci lokální sítě. Proto je potřeba upravit funkci obsluhující přijaté požadavky tak, aby odchytila specifické url adresy pro účel konfigurace, i pokud byly tyto požadavky přijaty přes TLS spojení. Pokud by webové prohlížeče přistoupily k blokování webových stránek, které jsou z části přijaté protokolem HTTP a z části protokolem HTTPS, pak by tato úprava



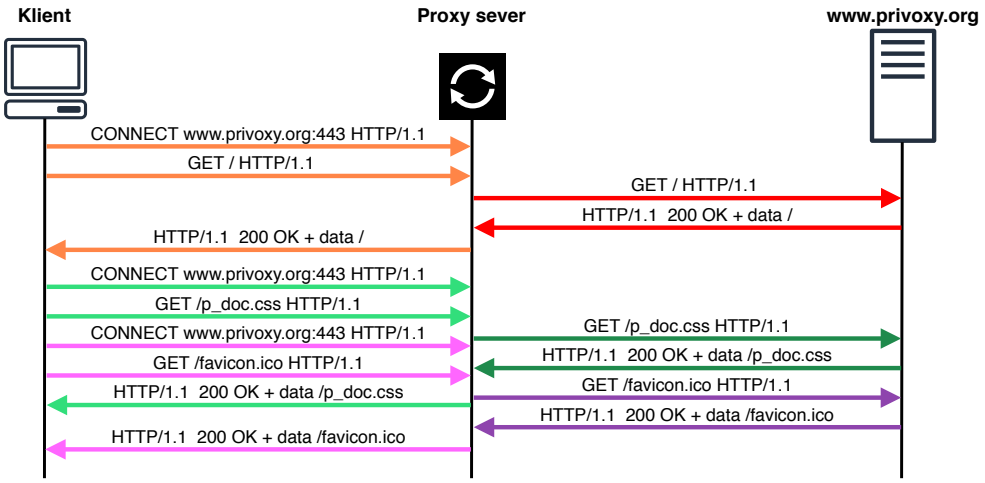
Obrázek 3.1: Srovnání doby načítání vybraných webových stránek podle použité metody a počtu jejích subresources. Použitý proxy server je výstupem předchozí bakalářské práce.

předešla blokování konfiguračního rozhraní. K tomu by mohlo dojít, jelikož výchozí url adresa pro konfiguraci je subdoménou `www.privoxy.org`.

3.3.6 Opakované používání TLS sessions

Pokud srovnáme dobu potřebnou pro načtení kompletního obsahu webové stránky bez použití proxy serveru, s použitím původního řešení implementovaného v Privoxy (TLS tunel) a s metodou MITM implementovanou v rámci bakalářské práce viz obrázek 3.1, vidíme velký rozdíl především u metody MITM. Při použití metody MITM se doba načítání stránky značně prodlužuje, a to především s rostoucím počtem subresources, tedy pokud se webová stránka skládá z velkého množství dalších prvků, které je potřeba samostatně načítat.

Důvod tohoto nárůstu doby načítání můžeme vidět na obrázku 3.2. Ten znázorňuje průběh načítání webové stránky `www.privoxy.org` s použitím proxy serveru, který využívá metodu MITM k filtrování komunikace. Jedná se tedy o přístup implementovaný v rámci bakalářské práce. Klient nejdříve zašle požadavek na adresář „/“, na který mu server vrátí indexový soubor. Následně klient zasílá požadavky na další části stránky odkazované z indexového souboru, tedy na soubory „p_doc.css“ a „favicon.ico“. Pro každý z těchto dotazů vytváří proxy server nové TLS spojení s klientem i webovým serverem. Tato spojení jsou na obrázku 3.2 odlišena různými barvami šipek. Celkem je tedy vytvářeno šest samostatných TLS spojení, která jsou ihned po přenesení souboru uzavírána. Režie na vytváření TLS spojení zpomaluje načítání webové stránky a nese tak nezanedbatelný podíl na celkové době potřebné pro její načtení. Opakované využití vytvořených TLS spojení by tak urychlilo načítání, zejména pro webové stránky s velkým množstvím subresources. Původní verze



Obrázek 3.2: Načítání jednotlivých částí webové stránky `www.privoxy.org` s použitím proxy serveru Privoxy s rozšířeními implementovanými v rámci bakalářské práce.

Privoxy již umožňuje zachovávat navázaná TCP spojení a následně je i opakovaně využít. Pro tento účel zohledňuje hodnotu parametru `Keep-Alive` z HTTP hlavičky. Zachovávání vytvořených TLS spojení ovšem Privoxy neumožňuje.

K opakovanému využití TLS spojení se využívají tzv. TLS session. Ty umožňují obnovit dříve vytvořená TLS spojení pomocí session identifikátoru, který obě komunikující strany vyjednaly během TLS handshake. Při opětovném navazování TLS spojení může klient tento identifikátor na server zaslat, a díky tomu zrychleně obnovit dřívější předchozí TLS. Server ovšem může přijatý identifikátor odmítnout jako neplatný.[8] Opakované využití TLS sessions může urychlit načítání webové stránky, i pokud jsou její zdroje na různých webových serverech, avšak pouze pokud alespoň některé z těchto zdrojů sdílí společný webový server.

3.3.7 Sdílení TLS sessions mezi jednotlivými klienty

Původní proxy server Privoxy umožňuje TCP spojení nejen uchovávat k opětovnému použití. K dispozici je i funkcionalita umožňující existující TCP spojení sdílet mezi jednotlivými klienty, respektive mezi vlákny zpracovávající jednotlivá spojení ze strany klienta. Pokud tedy jeden klient zašle požadavek například na `www.privoxy.org`, proxy server jej odbaví a uchová spojení k dalšímu použití. Pokud následně v určitém časovém intervalu přijde další požadavek od jiného klienta na stejnou doménu, proxy server nenavazuje nové spo-

jení, ale poskytne již existující spojení novému klientovi. Tím dochází k úspoře času potřebného pro navázání nového spojení. Tento princip je vhodné aplikovat i pro vytvořená TLS spojení a za určitých okolností tak urychlit načítání webové stránky.

Sdílení spojení mezi klienty má ovšem důsledky nejen pro rychlost, ale i pro bezpečnost. Pokud proxy server využívá několik uživatelů, mohou jednotliví uživatelé používat i spojení jiných uživatelů. To může být nebezpečné, pokud jsou použita autentizační schémata jako je NTLM, kde je autentizováno pouze spojení jako celek a ne jednotlivé dotazy.[41] Zároveň tento přístup může oslabovat další metody zabezpečení, které využívají informace o konkrétním spojení. Jako příklad můžeme uvést metodu Token Binding, která kryptograficky svazuje aplikační tokeny s TLS vrstvou. Z těchto důvodů by měl správce proxy serveru povolit sdílení TLS sessions pouze v případě, že si je vědom veškerých souvisejících rizik a zvážil je oproti potencionálním výhodám tohoto řešení. K povolení sdílení TLS sessions slouží klíčové slovo `connection-sharing` v konfiguračním souboru `config`. Toto klíčové slovo sloužilo již v předchozích verzích Privoxy pro sdílení TCP spojení.

3.3.8 Konfigurace šifrových sad

Kryptografické knihovny při navazování TLS spojení odesílají protější straně šifrovou sadu, pomocí které specifikují podporované algoritmy pro výměnu klíče, šifrování pomocí tohoto klíče a pro ověření zpráv. Seznam je navíc seřazený sestupně podle preferencí. Protější strana porovná přijatou sadu s vlastními podporovanými algoritmy, vybere z nich nejvhodnější a odešle tento návrh klientovi. Klient návrh potvrdí nebo zamítne. V původní verzi Privoxy nelze zvolit vlastní šifrovou sadu, a tak knihovna použije výchozí, která je specifická pro každou verzi knihovny. Výchozí šifrová sada je jakýmsi kompromisem mezi bezpečností a kompatibilitou. V knihovně LibreSSL je ovšem v této sadě povolen i algoritmus RC4, který již není považován za bezpečný. Pro zvýšení úrovně zabezpečení je tedy vhodné, aby mohl klient sám definovat konkrétní šifrovou sadu pro vybrané webové servery. Díky tomu může nejen nastavit vysokou úroveň zabezpečení, ale z důvodu zpětné kompatibility může udělit vybraným serverům i výjimku.

Kapitola 4

Implementace

Součástí diplomové práce je i implementace navržených úprav. Implementace navazuje na výstup předchozí bakalářce práce, kterou dále rozšiřuje. Mnohá z těchto rozšíření je možné implementovat řadou odlišných způsobů, které následně ovlivňují funkčnost, bezpečnost i rozšiřitelnost výsledného řešení. Proto tato kapitola popisuje principy zvoleného řešení, včetně důvodů pro použití konkrétních postupů.

4.1 Použití verzovacího systému Git

Pro implementaci diplomové práce byl vytvořen repozitář na serveru GitLab. Výchozí commit obsahuje zdrojový kód Privoxy verze 3.0.28, která je nyní nejaktuálnější verzí tohoto proxy serveru. Zdrojový kód bakalářské práce ovšem vychází z Privoxy verze 3.0.26. Hlavní změny mezi těmito verzemi, které mají největší dopad na spojení této verze s bakalářskou prací, jsou:

- Rozdělení funkce `Chat`

  Funkce `Chat` je volána po akceptování připojení od klienta. Jejím úkolem je přijmout požadavek klienta, zpracovat ho, přeposlat na cílový server a přijatou odpověď předat klientovi. Ve starších verzích byla tato funkce velmi rozsáhlá a nepřehledná, a tak došlo k jejímu rozdělení do dvou samostatných funkcí. První část zůstala součástí funkce `chat` a zajišťuje načtení HTTP hlavičky od klienta, její parsování a nastavení parametrů spojení podle jejího obsahu. Druhá funkce `handle_established_connection` obsahuje cyklus pro zpracování, filtrování a přeposílání dalších přenášených dat. Zároveň došlo v nové verzi k celkovému přepracování zdrojového kódu funkce `chat`, které přineslo například zredukování duplicit díky vytvoření nových funkcí pro práci s daty. Změny implementované v rámci bakalářské práce zasahují zejména do funkce `chat`, a proto bylo nezbytné přizpůsobit zdrojový kód bakalářské práce nové verzi Privoxy.

- Nahrazení funkce `select` funkcí `poll`

  Původní verze Privoxy využívá k určení aktivních soketů funkci `select`. Nová verze ovšem k tomuto účelu nabízí i funkci `poll` a oznamuje, že v budoucnu tato funkce zcela nahradí funkci `select`. Důvodem změny je zejména problematické škálování počtu soketů při použití funkce `select`.[42] To je způsobeno souvisejícím makrem `fd_set` pro vložení soketu do množiny, které používá bitovou masku určité omezené délky. Naproti tomu funkce `poll` umožňuje uživateli alokovat pole soketů a předat jeho velikost, tudíž neomezuje počet soketů.[43]

Z těchto důvodů bylo nezbytné před implementací nových funkcionalit přizpůsobit zdrojový kód bakalářské práce nové verzi Privoxy. Proto byl nejprve zdrojový kód bakalářské práce rozdělen na menší logické celky, které lze snadněji přizpůsobit změnám v novější verzi Privoxy. Tyto celky pak byly samostatně připojovány k úvodnímu commitu. Nejvýznamnějšími částmi jsou:

1. Nahrazení metody TLS tunelu metodou MITM

   - Vytváření TLS spojení s cílovým serverem

     Na základě klientova požadavku `CONNECT`, přijatého přes nezabezpečené spojení, iniciuje proxy server navazování zabezpečeného spojení s požadovaným webovým serverem.
   - Vytváření TLS spojení s klientem

     Poté, co proxy server úspěšně naváže TLS spojení s webovým serverem, potvrdí tuto skutečnost klientovi, a následně s ním naváže zabezpečené TLS spojení.
   - Přeposílání dat mezi klientem a serverem

     Poté co je navázáno TLS spojení se serverem i s klientem, je potřeba přeposílat data mezi nimi. Jako první je serveru zaslán požadavek klienta. Následně jsou v cyklu přeposílána veškerá data, která proxy server přijme. Přijetí dat je detekováno pomocí funkce `select`, případně jsou data čekající k načtení na zásobníku ověřena pomocí rozhraní kryptografické knihovny. Veškeré úpravy musí zachovávat funkčnost i pro nezabezpečená spojení.
   - Použití nadřazeného proxy serveru

     Při použití nadřazeného proxy serveru je potřeba celý postup upravit. Proxy server nenavazuje TLS spojení s cílovým serverem, ale přeposílá požadavek `CONNECT` od klienta na nadřazený proxy server. Zároveň proxy sever nevytváří potvrzení o navázání spojení, ale pouze přeposílá potvrzení vytvořené nadřazeným proxy serverem.
   - Podpora funkce `poll`

     Pokud klient podporuje funkci `poll`, je použití této funkce preferováno. Bylo tedy nezbytné přizpůsobit detekci příchozích dat

pomocí funkce `poll` i při použití TLS spojení. Proto je v případě čekajících dat na zásobníku ručně nastavena událost `POLLIN` signalizující tuto skutečnost, která je dále standardně zpracována.

2. Zpracování certifikátů

   - Kontrola platnosti certifikátů

     Během navazování TLS spojení s požadovaným serverem musí proxy server ověřit platnost přijatého certifikátu. Pokud je certifikát platný, komunikace může pokračovat. V opačném případě je klient informován o důvodech, proč nebyl certifikát uznán jako platný.

   - Vytváření certifikátů webových stránek

     Proxy server musí webovému prohlížeči v průběhu TLS handshake zaslat certifikát korespondující s požadovanou doménou. V opačném případě by webový prohlížeč přerušil navazování spojení a informoval uživatele o příčinách. Proto musí proxy server generovat vlastní soukromé klíče a pomocí vlastní CA podepisovat pro každou z požadovaných domén odpovídající certifikát. Aby nebyl certifikát vytvářen opakovaně, jsou vytvořené certifikáty ukládány a následně načítány jednotlivými vlákny, která obsluhují jednotlivé klienty. Z důvodu paralelního běhu ve více vláknech jsou využity mutexy pro zabránění kolizím.

3. Konfigurace

   Součástí bakalářské práce je i rozšíření možností konfigurace. Konkrétně se jedná o konfiguraci parametrů pro generování certifikátů, dále je umožněno pro zvolené url adresy nahradit nově implementovanou metodu MITM původním řešením pomocí TLS tunelu a poslední rozšíření konfigurace umožňuje pro vybrané url adresy vypnout kontrolu certifikátu webového severu. Všechna tato rozšíření byla rovněž přizpůsobena nejnovější verzi Privoxy.

4.2 Změna kryptografické knihovny

Hlavním důvodem pro změnu kryptografické knihovny je zachování kompatibility s moderními webovými prohlížeči. Privoxy je licencováno pod licencí GNU GPLv2[8]. Je tedy nezbytné zvolit takovou kryptografickou knihovnu, která nejen že nabídne veškeré potřebné funkcionality, ale zároveň bude kompatibilní s touto licencí. Proto byla v rámci diplomové práce použita kryptografická knihovna LibreSSL[9]. Tato knihovna vznikla v roce 2014 odštěpením

[8] GNU General Public License, version 2. Dostupné na: `https://www.gnu.org/licenses/old-licenses/gpl-2.0.html`

[9] Kryptografická knihovna LibreSSL. Dostupné na: `https://www.libressl.org/`

od knihovny OpenSSL s cílem modernizovat původní zdrojové kódy a zlepšit bezpečnost. Knihovna je licencována pod licencí OpenBSD.[10]

Kryptografická knihovna LibreSSL umožňuje nastavovat rozšíření `Subject Alternative Name` u vytvářených certifikátů, které je nezbytné pro zajištění funkčnosti s novými verzemi webových prohlížečů. Zároveň tato knihovna nabízí i podporu protokolu APLN, který je využíván při navazování HTTP/2 spojení. Díky tomu by mohla být tato knihovna využita, i pokud by se Privoxy v budoucnu rozhodlo pro podporu tohoto protokolu. Jedním z nedostatků této knihovny je aktuálně chybějící podpora CT.

Kvůli změně kryptografické knihovny bylo následně nutné implementovat veškeré kryptografické funkcionality, které již byly implementovány v předchozí bakalářské práci. Jedná se o funkce pro navazování TLS spojení, přeposílání dat přes TLS spojení, kontrola webových certifikátů, vytváření webových certifikátů a další funkce, z nichž některé jsou detailněji popsány v části 4.1.

4.3 Konfigurace kompilačních skriptů

Díky použití nové kryptografické knihovny byla vytvořena druhá funkční verze proxy serveru Privoxy implementující princip MITM. Každá z těchto verzí má své výhody i nevýhody a jednotliví uživatelé mohou preferovat jinou z těchto dvou knihoven. Zároveň mohou někteří uživatelé preferovat používání Privoxy bez možnosti filtrovat HTTPS komunikaci a tedy i bez využívání jakékoli kryptografické knihovny. Z těchto důvodů byl v rámci diplomové práce upraven soubor `configure.in`, na jehož základě generuje nástroj `Autoconf` konfigurační skript. S využitím tohoto konfiguračního skriptu jsou následně vytvořeny soubory `makefile` a `config.h`. Zatímco `makefile` slouží ke kompilaci zdrojových kódů, soubor `config.h` obsahuje direktivy `#define` definující jednotlivé identifikátory. Tyto identifikátory jsou následně využívány v rámci zdrojových kódů u direktiv `#ifdef` pro podmíněné kompilování bloků zdrojového kódu.

Aby mohli uživatelé zvolit preferovanou kryptografickou knihovnu, která bude použita v průběhu kompilace, byl soubor `configure.in` rozšířen o dva nové parametry. Ty mohou být použity při volání konfiguračního skriptu. Způsob použití těchto parametrů je popsaný v tabulce 4.1, ve které je uvedena výsledná kryptografická knihovna, která bude při zvolené kombinaci přepínačů použita v průběhu kompilace. Na základě použitých přepínačů jsou konfiguračním skriptem upraveny direktivy `#define` jednotlivých identifikátorů v souboru `config.h`, a zároveň jsou nastaveny proměnné v `makefile` souboru. Tyto identifikátory následně aktivují či deaktivují příslušné bloky zdrojového kódu v průběhu kompilace. Uživatel následně použije příkaz `make` k provedení kompilace podle svých preferencí.

[10] OpenBSD Copyright Policy. Dostupné na: `https://www.openbsd.org/policy.html`

Tabulka 4.1: Použití nových přepínačů konfiguračního skriptu

	--disable-libressl	--enable-mbedtls	Žádný přepínač
--disable -libressl	Bez kryptografické knihovny	Mbed TLS	Bez kryptografické knihovny
--enable -mbedtls	Mbed TLS	Mbed TLS	Mbed TLS
Žádný přepínač	Bez kryptografické knihovny	Mbed TLS	LibreSSL

4.4 Filtrování přijatých požadavků

Při použití HTTPS spojení umožňuje předchozí bakalářská práce filtrovat pakety přijaté od webového serveru včetně jejich HTTP hlaviček. Práce ovšem umožňuje jen částečné filtrování HTTP hlaviček u příchozích požadavků, pokud jsou přijaty přes HTTPS spojení. V bakalářské práci totiž zůstal zachován bez větších úprav mechanismus zpracovávající přijaté požadavky od klienta. Tento mechanismus umožňuje filtrovat veškeré HTTP hlavičky přijaté přes nezabezpečené spojení, ale při použití HTTPS se komunikace s klientem liší a pro filtrování všech přijatých hlaviček jsou nutné rozsáhlejší změny.

4.4.1 Zpracovaní HTTP požadavku samostatnou funkcí

Původní implementace Privoxy již obsahuje několik funkcí, které při postupném volání umožňují přijmout a zpracovat HTTP hlavičku klientova požadavku. Dvě nejdůležitější funkce pro přijetí požadavku jsou:

- `receive_client_request`

  Funkce zabezpečuje přijímání dat od klienta tak, aby byl přijat vždy celý požadavek. Následně provádí kontrolu přijatého požadavku. Protokol přijatého požadavku musí být proxy serverem podporován a struktura požadavku musí odpovídat použitému protokolu. Následně je první řádka HTTP požadavku rozdělena na jednotlivé parametry jako je verze protokolu, metoda požadavku, url adresa a port. Zbytek požadavku je rozdělen po jednotlivých řádkách.

- `parse_client_request`

  Funkce podle přijatého požadavku nastavuje hodnoty proměnných, s jejichž pomocí se řídí průběh zpracování požadavku i odpovědi na něj. Následně na tento požadavek aplikuje příslušné filtry, které umožňují upravit jeho HTTP hlavičku i obsah.

V původní verzi Privoxy docházelo k přijetí a zpracování HTTP požadavku pouze jednou pro každý požadavek klienta, tedy pouze na jednom místě v kódu. Jelikož pro účel filtrování HTTPS komunikace je nezbytné načtení a parsování požadavku opakovaně, byla vytvořena nová funkce `process_client_request`. Tato funkce sdružuje veškeré funkce pro zpracování požadavku klienta do jednoho celku. Voláním této funkce lze opakovaně načítat požadavky klienta. Zároveň je možné upravit a doplnit původní funkce na jednom jediném místě o další rozšíření a předchází se i duplicitám v kódu. Nahrazení rozsáhlejších bloků kódu z původní funkce `chat` navíc tuto funkci výrazně zkrátí a zvýší tak její přehlednost.

4.4.2 Zapouzdření funkcí pro zpracování dat

Existující funkce pro přijímání (respektive kontrolu dostupnosti) a odesílání dat jsou v originální verzi Privoxy implementovány pouze pro TCP spojení. V bakalářské práci pak byly doplněny ekvivalentními funkcemi pro TLS spojení. Privoxy zároveň obsahuje řadu funkcí pro zpracování přijatých dat, které jsou přizpůsobeny pouze pro TCP spojení. V bakalářské práci byly některé z těchto funkcích upraveny pro potřeby TLS spojení. Pro všechna volání funkcí závisejících na typu použitého spojení tak byla vložena `if-else` podmínka, která podle typu vytvořeného spojení volá funkci obsluhující toto spojení. Pro každé přijímání či odesílání dat tedy byla vytvořena podmínka typu:

```
if (client_use_ssl(csp)) {
    ret = ssl_send_data(&(csp->ssl_client_attr.ssl),
              receive_buffer, len);
    if (ret < 0) {
        ...
    }
} else {
    ret = write_socket(csp->cfd, receive_buffer, len)
    if (ret < 0) {
        ...
    }
}
```

Pro filtrování požadavků přijatých přes zabezpečené TLS spojení je potřeba využít některé již existující funkce. Tyto funkce ovšem musí být přizpůsobeny jak pro TCP spojení, tak pro zabezpečená TLS spojení. Aby nebylo nutné nahrazovat všechna volání funkcí pro komunikaci výše uvedenou podmínkou, byly vytvořeny nové funkce pro každou z operací nad spojením. Tyto funkce přijímají strukturu `client_state`, která obsahuje všechny potřebné proměnné pro komunikaci s klientem či serverem přes existující spojení s ním, bez ohledu na typ tohoto spojení. Dále funkce přijímají parametr typu `int`, pro který jsou předdefinována makra `CLIENT` a `SERVER`, případně další potřebné parametry

Tabulka 4.2: Výběr obsluhovaného spojení na základě parametrů

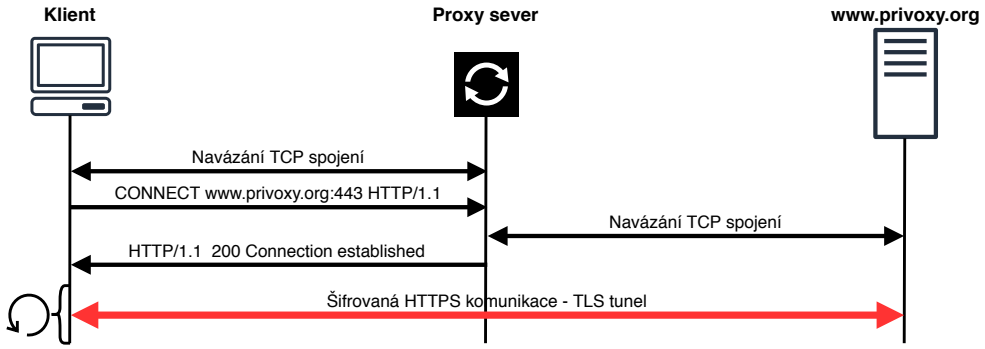
		Cíl komunikace	
		Klient	Server
Typ spojení	TCP	Komunikace s klientem přes nezabezpečené spojení	Komunikace se serverem přes nezabezpečené spojení
	TLS	Komunikace s klientem přes zabezpečené TLS spojení	Komunikace se serverem přes zabezpečené TLS spojení

jako je zásobník pro data a jeho délka. Volající tak funkci předá potřebné struktury a definuje, jestli má funkce číst data od serveru či klienta. Volaná funkce pak sama na základě parametrů vybere vhodnou funkci a předá jí potřebné parametry. Všechny varianty, které funkce bere v úvahu, jsou zobrazeny v tabulce 4.2.

Díky zapouzdření všech původních funkcí pro komunikaci mohly být původní `if-else` bloky ve zdrojovém kódu nahrazeny těmito novými funkcemi. To následně umožnilo bez rozsáhlejších úprav využít již existující funkce v Privoxy pro zpracování nezabezpečené komunikace, i pokud se jedná o zabezpečená spojení. Použité spojení je zvoleno automaticky na základě předaných parametrů.

4.4.3 Přijetí HTTP hlavičky přes zabezpečené spojení

Původní verze proxy serveru Privoxy využívá pro zpracování HTTPS komunikace tzv. tunel. Tato metoda je znázorněna na obrázku 4.1. Pokud klient ví, že jeho komunikace je směrována na proxy server (tuto informaci získá z nastavení webového prohlížeče, případně z nastavení operačního systému), zahajuje komunikaci požadavkem typu `CONNECT`. V něm uvádí používanou verzi protokolu HTTP a požadovanou url adresu. Ta může obsahovat nejen cílovou doménu, ale i požadovaný protokol, případně port. Na základě těchto informací se proxy server pokusí navázat TCP spojení s cílovým serverem. Pokud je použit nadřazený proxy server, aplikují se na požadavek `CONNECT` definované filtry a následně je přeposlán nadřazenému proxy serveru. Po úspěšném navázání TCP spojení s cílovým serverem (respektive poté, co spojení s cílovým serverem naváže nadřazený proxy server) je o tom klient informován. Veškerá následující komunikace mezi klientem a webovým serverem je proxy serverem již pouze přeposílána ve znázorněné smyčce. Pokud klient vyžaduje zabezpečené spojení, vytváří toto spojení přímo s webovým serverem. Po navázání TLS spojení jsou již veškerá přenášená data šifrována, a tak na ně nemůže proxy server aplikovat filtrování. To se týká i následně zaslaného

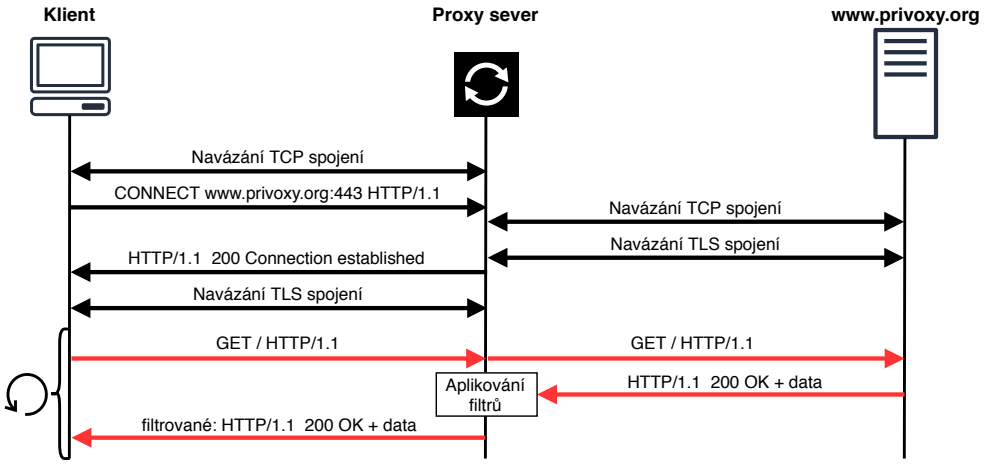


Obrázek 4.1: Zpracování HTTPS komunikace původním proxy serverem Privoxy při použití metody TLS tunelu. Černé šipky znázorňují TCP spojení, červená zabezpečené TLS spojení.

požadavku typu `GET`, kterým klient serveru specifikuje požadovaná data.

Bakalářská práce tento přístup částečně upravila. Nahradila TLS tunel metodou MITM. Princip této metody, tak jak byl implementovaný v bakalářské práci, je znázorněný na obrázku 4.2. Proxy server po přijetí požadavku `CONNECT` navazuje nejprve TCP spojení a následně i zabezpečené TLS spojení s požadovaným webovým serverem. Poté informuje klienta o úspěšném vytvoření tohoto spojení. Když se klient následně pokouší navázat TLS spojení s webovým serverem, proxy server sám vystupuje jako webový server a naváže s klientem TLS spojení. Poté klient zašle požadavek typu `GET` na konkrétní soubor. Proxy server už ovšem stejně jako u metody TLS tunelu ve smyčce okamžitě přeposílá tento požadavek webovému serveru. Neaplikují se na něj žádné filtry, jelikož původní proxy server Privoxy počítal s přijetím pouze jediného HTTP požadavku, který mohl být zpracován. Při použití TLS tunelu již žádný další požadavek nebyl pro proxy server čitelný a nemohl jej tak zpracovat. Při použití nezabezpečeného HTTP spojení klient nezasílá požadavek typu `CONNECT`, ale již první požadavek je typu `GET`. Ten obsahuje i url adresu cílového serveru. Žádný další požadavek není proxy serverem přijat. Původní verze Privoxy již implementovala filtrování dat přenášených od webového serveru, pokud byla přenášena pomocí nezabezpečeného spojení. V bakalářské práci pak byla tato funkcionalita rozšířena i na zabezpečená spojení, jelikož nebyly potřebné rozsáhlejší úpravy.

Aby mohl být filtrován i druhý požadavek zaslaný klientem při použití zabezpečeného spojení, je potřeba jej nejdříve načíst do připravených struktur. Proto byla funkce `chat` doplněna o druhé volání funkce `process_client_request` po navázání TLS spojení s klientem. Tak je zpracován i druhý HTTP požadavek od klienta. Díky předchozím úpravám funkce sama zvolí používaný

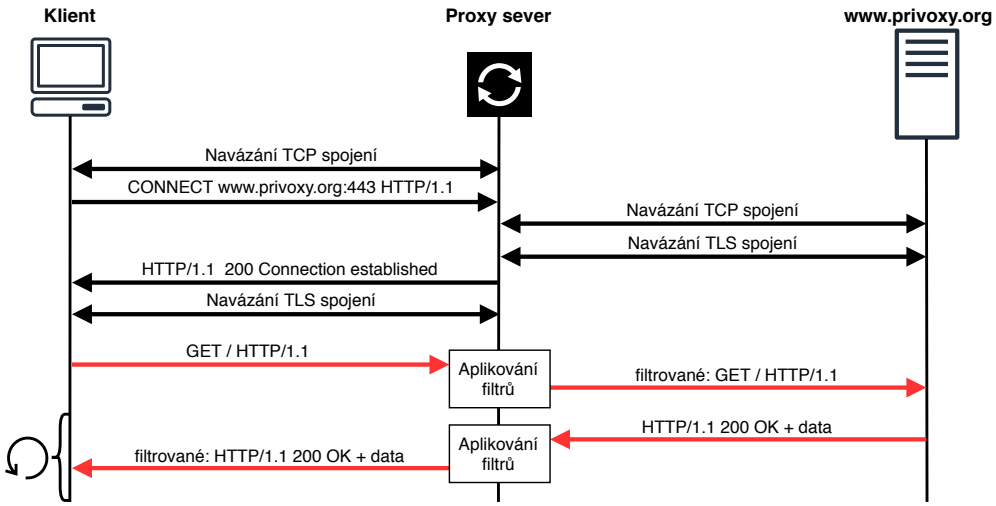


Obrázek 4.2: Zpracování HTTPS komunikace proxy serverem Privoxy s rozšířením implementovaným v bakalářské práci (použití metody MITM). Černé šipky znázorňují TCP spojení, červené zabezpečená TLS spojení.

typ spojení. V tomto případě tedy spojení typu TLS. Hlavička načteného požadavku je vkládána do totožné struktury jako předchozí požadavek typu `CONNECT`. Funkce ale v případě načítání druhého požadavku v této struktuře jen doplní či přepíše některé z původních hodnot v závislosti na jejich typu. Parametry jako je použitý port a adresa cílového serveru jsou beze změny uchovány, jelikož je nový požadavek neobsahuje. Obsahuje naopak relativní cestu na požadovaný soubor, tak jak je znázorněno na obrázku 4.2. Upravení předchozího požadavku typu `CONNECT` je možné, protože některé jeho parametry již nejsou potřeba, případně je druhý požadavek ani neobsahuje, a tak díky úpravě funkce `get_destination_from_headers` nejsou přepsány. Tento přístup umožňuje využít původní funkce pro načítání požadavku bez rozsáhlejších úprav. Na takto zpracovaný požadavek jsou aplikovány odpovídající filtry a požadavek je přeposlán webovému serveru přes existující zabezpečené spojení. Teprve poté je spuštěna funkce `handle_established_connection`, která v cyklu přeposílá data přijatá od serveru a filtruje jejich obsah. Kompletní postup je zobrazen na obrázku 4.3.

4.4.4 Další zjednodušení a opravy

Aby byla i nadále zachována jednoduchost a přehlednost zdrojového kódu, byly původní rozsáhlé funkce rozděleny do menších logických celků. Jednou z takových úprav je vytvoření funkce `create_server_connection` pro navázání spojení s webovým serverem. Tento celek byl původně součástí funkce `chat`.



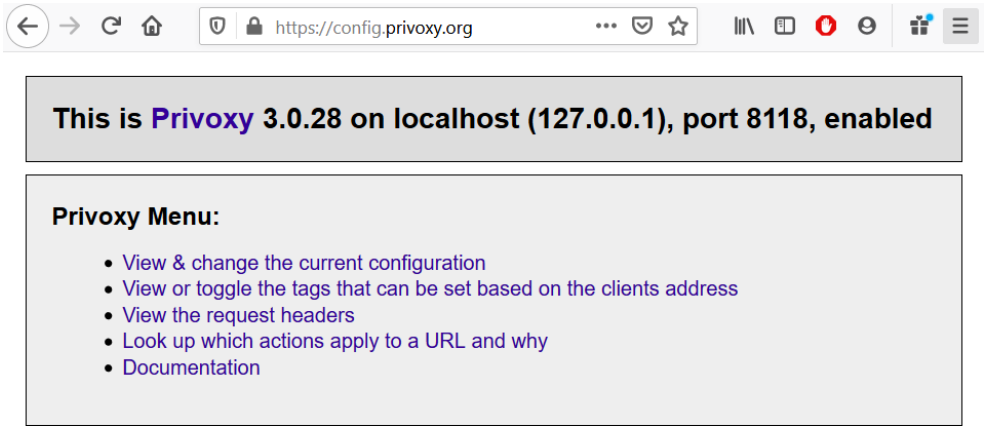
Obrázek 4.3: Zpracování HTTPS komunikace proxy serverem Privoxy s rozšířením implementovaným v diplomové práci. Černé šipky znázorňují TCP spojení, červené zabezpečené TLS spojení.

Funkce obsahuje nejen samotné navázání TCP a TLS spojení, ale i veškerá s nimi související nastavení příslušných proměnných, postup pro navázání spojení s nadřazeným proxy serverem, postup pro vytvoření TLS tunelu, či informování klienta o neplatnosti certifikátu cílového serveru.

Nejen díky těmto zjednodušením se podařilo v rámci diplomové práce odhalit i chybu v původní verzi Privoxy 3.0.28. Spočívá v používání neinicializované proměnné `csp->fwd` ve funkci `handle_established_connection`. Tato proměnná by měla obsahovat parametry nadřazeného proxy serveru, pokud je použit. Ve verzi 3.0.26 již nebyla tato proměnná inicializována, nebyla však ani používána a místo ní se příslušné parametry vkládaly do lokální proměnné `fwd`. Ve verzi 3.0.28 byla funkce `chat` obsahující zmíněnou lokální proměnnou rozdělena na dvě samostatné funkce. V nové funkci bylo použití proměnné `fwd` nahrazeno proměnnou `csp->fwd`, jelikož `csp` získá jako parametr. Proměnná `csp->fwd` ovšem stále zůstala neinicializována. Tato chyba byla opravena.

4.5 Změny konfiguračního rozhraní

Aby mohlo být existující konfigurační rozhraní pro klienta dostupné i pomocí zabezpečeného spojení (obrázek 4.4), je nezbytné rozšířit již existující funkce, které toto rozhraní poskytují pouze přes nezabezpečené spojení. Zároveň je nutné doplnit algoritmus pro zpracování požadavku klienta o nové možnosti.



Obrázek 4.4: Část konfiguračního rozhraní Privoxy získaná pomocí zabezpečeného spojení.

4.5.1 Odchycení HTTPS požadavků na konfigurační rozhraní

Po přijetí požadavku od klienta proxy server kontroluje, jestli má být tento požadavek zpracován přímo proxy serverem, nebo jestli má být přeposlán na webový server. Jelikož původní proxy server umožňuje odchytit a zpracovat pouze požadavky přijaté přes nezabezpečené spojení, může okamžitě generovat odpověď na základě požadavku. První přijatý požadavek při použití protokolu HTTPS je ovšem typu `CONNECT`, a tak z něj lze získat pouze požadovanou doménu, protokol a port. Proto může proxy server pouze detekovat skutečnost, že dotaz nebude přeposílán ale bude zodpovězen přímo proxy serverem. Kompletní informace získá ovšem až z druhého dotazu přijatého přes TLS spojení.

Kvůli zmíněným rozdílům mezi zabezpečeným a nezabezpečeným spojením musel být upraven původní způsob zpracování požadavků. Pokud proxy sever na základě požadavku zjistí, že pro požadovanou doménu bude generovat odpověď on sám, postupuje následovně podle typu spojení:

- Nezabezpečené spojení

  Odpověď je vygenerována a odeslána klientovi okamžitě, jelikož veškeré potřebné informace jsou k dispozici.

- Zabezpečené spojení

  Proxy server pokračuje obvyklým způsobem s tím rozdílem, že se nepokouší navázat spojení s požadovaným webovým serverem. Klientovi ovšem odešle zprávu `200 Connection established`. Následně klient zasílá druhý požadavek, kterým specifikuje zbývající parametry. Na jejich základě může proxy sever vygenerovat odpověď a odeslat ji klientovi.

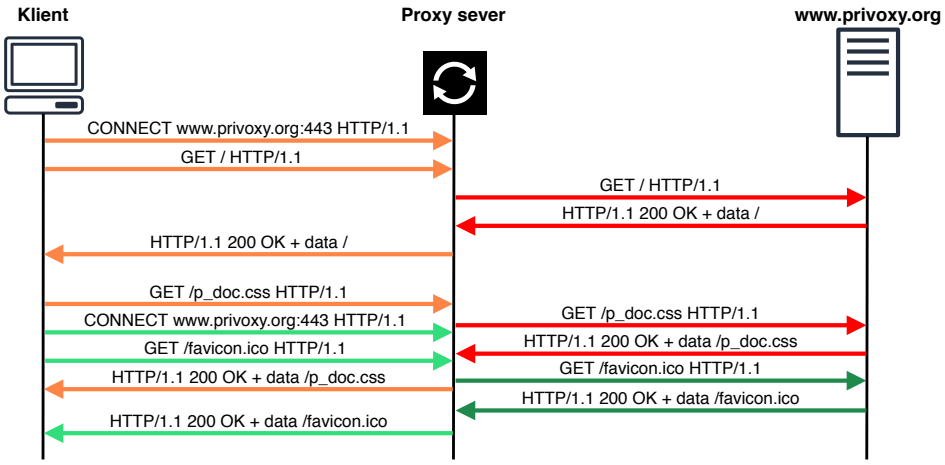
Původní funkce `crunch_response_triggered` pro rozhodnutí o odchycení požadavku, generování a odeslání odpovědi byla doplněna funkcí `should_trigger_crunch_response`. Ta rozhodne o odchycení požadavku, a na základě parametru `prepare_rsp` vygeneruje konkrétní odpověď na požadavek. Tato funkce je pak volána buďto samostatně, nebo z původní funkce `crunch_response_triggered`, jejíž chování je zachováno, tedy vždy zároveň vygeneruje a odešle odpověď.

4.5.2 Změna šablon konfiguračních webových stránek

Pro korektní chování konfiguračního rozhraní ovšem nestačilo upravit postup zpracování požadavků. To samotné sice umožní získat pomocí zabezpečeného spojení webovou stránku vytvořenou proxy serverem. Pokud by ovšem odkazy v těchto stránkách i nadále odkazovaly na nezabezpečená spojení, uživatel by při první použití libovolného odkazu přešel opět na nezabezpečené spojení. Proto musely být upraveny i veškeré funkce vytvářející obsah webových stránek. Pro tento účel byly funkce doplněny o nový parametr, na jehož základě se funkce dozví, přes jaký typ spojení bude vygenerovaný obsah přenášen, a tedy jaký protokol má být specifikovaný u jednotlivých odkazů. Odkazy tak doplní prefixem „`http://`“ nebo „`https://`“.

4.6 Opakované používání TLS sessions

Aby nedocházelo ke zbytečnému navazování a ukončování TLS spojení mezi vyřizováním jednotlivých požadavků klienta, je nutné tato spojení uchovat i po odbavení požadavku, pro který byla vytvořena. Díky tomu může být spojení s klientem využito i pro přijetí dalšího požadavku na totožný webový server, a následně lze znovu využít i existující spojení s webovým serverem. Výsledný efekt je znázorněn na obrázku 4.5. Oproti původnímu řešení bakalářské práce zobrazenému na obrázku 3.2 je patrné, že při opakovaném používání TLS spojení jsou místo původních šesti TLS spojení vytvářena pouze čtyři (uvedené hodnoty platí pro konkrétní webovou stránku `www.privoxy.org`). Nejdříve je od klienta přijat požadavek `CONNECT`. Podle něj proxy server vytvoří první TLS spojení. Poté je s klientem navázáno druhé TLS spojení. Přes tato dvě spojení je vyřízen klientův požadavek na první dokument. Klient zpracuje přijatá data a vyžaduje další dva soubory z totožného webového serveru. První požadavek zasílá přes již existující TLS spojení, které bylo pro tento účel uchováno. Spojení je v ten okamžik plně využito, a pokud klient nepodporuje zasílání více požadavků před obdržením odpovědi na předchozí (výchozí chování většiny moderních webových prohlížečů), nemůže být v tu chvíli použito pro další požadavky. Zasílání několika požadavků před obdržením odpovědi na první z nich se nazývá pipelining, který proxy server Privoxy nepodporuje ani v nejnovější verzi (pouze umožňuje jeho tolerování, ale ani tuto možnost nedoporučuje). Klient tedy zasílá nový požadavek `CONNECT`, podle kterého proxy



Obrázek 4.5: Princip opakovaného používání zabezpečených spojení. Každá barva představuje samostatné TLS spojení.

server vytvoří nové TLS spojení (celkově již třetí). Následně je vytvořeno nové TLS spojení s klientem, přes které klient může zaslat požadavek na konkrétní dokument. Ten mu je obratem doručen. Celkem jsou tedy vytvořena čtyři TLS spojení pro získání tří souborů. Tato spojení zůstanou otevřená i po přenesení všech požadovaných dokumentů, a tak mohou být opětovně využita, dokud nevyprší časový limit `keep-alive-timeout` nastavený v konfiguračním souboru `config`. Díky tomu může klient následně odeslat dva současné požadavky na totožný webový server, aniž by se vytvářelo nové spojení. To přináší další zrychlení při vyřizování požadavků klienta.

4.6.1 Zvolený princip

Pro opakované využití TLS sessions je možné zvolit několik postupů. Zde jsou popsány dva možné přístupy, z nichž jeden byl implementován v rámci diplomové práce:

1. Obnova spojení pomocí session ID

   Během TLS handshake při vytváření nového spojení mezi klientem a webovým serverem jsou vyjednávány parametry session. Ty obsahují mimo kryptografických parametrů i tzv. session ID (případně session tiket či PSK v závislosti na verzi TLS). Pokud je následně toto TLS spojení ukončeno, může klient při dalším navazování TLS spojení využít toto session ID k obnovení předchozí session. Stačí, aby vložil na začátku TLS handshake do `Client Hello` zprávy session ID z předchozího spojení. Server následně ověří, že přijaté session ID odpovídá některé z před-

chozích sessions a rozhodne o přijetí či zamítnutí požadavku na obnovení session. Pokud je požadavek serverem přijat, jsou pro TLS spojení použity totožné kryptografické parametry jako v předchozím spojení a nemusí tak být prováděna velká část handshake. Tím dochází ke značné úspoře času pro navázání spojní. Pokud sever zamítne požadavek na obnovení session, proběhne klasický handshake.[6]

2. Uchování aktivního spojení

   Alternativním přístupem může být uchování spojení i po vyřízení požadavku klienta. Nedojde tak u uzavření TLS spojení. Spojení stále existuje, pouze není aktivně využíváno. K ukončení TLS spojení dochází pomocí upozornění typu `close_notify`. Tato zpráva může být zaslána libovolnou stranou a upozorní příjemce, že protější strana již nebude zasílat přes spojení žádná další data. Protější strana musí po obdržení `close_notify` zprávy odeslat totožnou zprávu protější straně. Není však nutné, aby protější strana před uzavřením spojení na tuto zprávu čekala. Dokud tedy není přijata `close_notify` zpráva, může být TLS spojení stále využíváno k zabezpečenému přenosu dat.[6]

Z těchto dvou metod byl v této diplomové práci implementován druhý z přístupů, který uchovává aktivní spojení. Ačkoli je jednou z jeho nevýhod například nutnost uchovávat větší množství dat v paměti proxy serveru oproti první metodě, zvolen byl zejména z následujících důvodů:

- Prodlevy mezi jednotlivými požadavky klienta na proxy server jsou velmi malé. Nejčastěji od několika desítek milisekund do desítek sekund. Spojení je tedy využíváno velmi často a nehrozí jeho ukončení ze strany webového serveru.
- Pokud není spojení ukončeno, dochází k ještě větší úspoře času při jeho opakovaném použití i v porovnání se zkráceným handshake.
- V porovnání s obnovou spojení pomocí session ID je toto řešení méně složité.

4.6.2 Správa stavu TLS spojení

Aby mohl být algoritmus pro zpracování dat od klienta i webového serveru přizpůsoben pro uchování aktivních spojení i pro další požadavky, je nezbytné mít v každém okamžiku přehled o stavu všech těchto spojení, respektive o tom, jak má být se spojeními zacházeno. Proto byla struktura `client_state`, která obsahuje veškeré informace vlákna zpracovávajícího požadavky klienta, rozšířena o proměnnou `ssl_flags`. Ta je využívána jako binární mapa, kde jednotlivé bity specifikují tyto vlastnosti:

- `0x01U = CSP_SSL_FLAG_SERVER_CONNECTION_REUSED`

  TLS spojení s webovým serverem (případně nadřazeným proxy serverem) je již využíváno opakovaně.

- `0x02U = CSP_SSL_FLAG_CLIENT_CONNECTION_REUSED`

  TLS spojení s klientem je již využíváno opakovaně.

- `0x04U = CSP_SSL_FLAG_CONNECTIONS_TO_BE_REUSED`

  TLS spojení s klientem i webovým serverem nemají být po vyřízení požadavku uzavřena, aby mohlo dojít k jejich opětovnému využití.

- `0x08U = CSP_SSL_FLAG_SERVER_CONNECTION_TAINTED`

  TLS spojení s webovým serverem musí být uzavřeno, jelikož může obsahovat neočekávaná data.

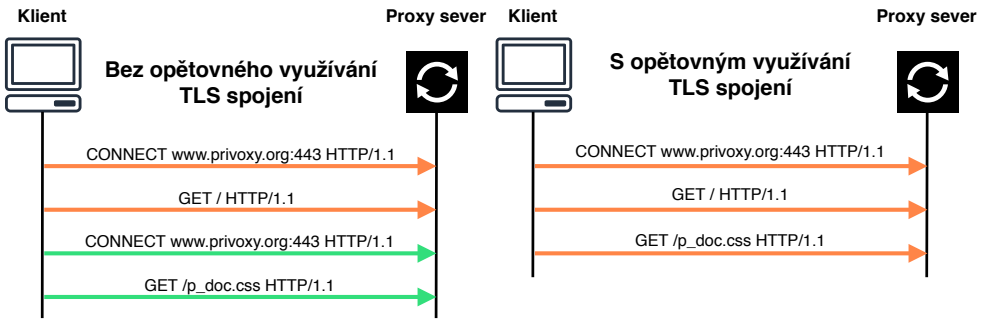
- `0x10U = CSP_SSL_FLAG_CLOSE_SERVER_CONNECTION`

  TLS spojení s webovým serverem musí být uzavřeno pro nespecifikovaný důvod.

4.6.3 Přijmutí požadavku přes opětovně použité spojení

Aby mohly být aplikovány filtry i na požadavky klienta přijaté přes zabezpečené spojení, byl upraven postup pro zpracování přijatých požadavků, jak je popsáno v sekci 4.4.3. Při opětovném využívání TLS spojení ovšem dochází k dalším změnám. První požadavek po navázání TCP spojení je typu `CONNECT`. V něm je definován cílový server a typ požadovaného spojení. Podle těchto parametrů proxy server vytvoří příslušné TLS spojení s cílovým serverem. Poté je vytvořeno TLS spojení i s klientem. Klient následně zasílá dotazy na jednotlivé prvky webové stránky. Pokud je ovšem TLS spojení využíváno opakovaně, není potřeba jej znovu vytvářet, a tudíž klient již nezasílá požadavek typu `CONNECT`, ale rovnou zašle požadavek na specifický soubor. Rozdíl je znázorněný na obrázku 4.6. Z obrázku je rovněž patrné, že požadavek typu `GET` na konkrétní soubor již neobsahuje celou url adresu, ale jen relativní cestu k požadovanému souboru.

Na základě uvedené proměnné `ssl_flags` tak byl upraven postup pro zpracování přijatých požadavků. Pokud proxy server přijímá první požadavek přes TCP spojení a jedná se o požadavek typu `CONNECT`, naváže požadované spojení a očekává ještě jeden požadavek na konkrétní soubor. Pokud je však požadavek přijatý přes již existující TLS spojení, očekává se, že již bude specifikovat konkrétní soubor. Žádný další požadavek již není až do vyřízení právě přijatého požadavku přijímán. Proxy server navíc i nadále podporuje původní postup, který je využíván, pokud správce proxy serveru požaduje pro vybrané url adresy použít TLS tunel.



Obrázek 4.6: Přijímání požadavků bez a s opětovným využíváním TLS spojení

4.6.4 Vytváření spojení s cílovým serverem

Funkce `create_server_connection` pro navázání spojení s požadovaným webovým serverem dosud vytvářela vždy nové TLS spojení, pokud ho klient vyžadoval. Veškeré parametry vytvořeného TLS spojení pak funkce vkládá do struktury typu `ssl_connection_attr`, která je uložena ve struktuře `client_state`. Tu obsahuje každé vlákno obsluhující konkrétního klienta.

Pokud má být TLS spojení opětovně využíváno, je potřeba nově vytvořené TLS spojení po vyřízení požadavku neuzavírat. Toho je dosaženo nastavením odpovídajících hodnot v proměnné `ssl_flags`. Ta je kontrolována při každém volání funkce pro uzavření TLS spojení. Pokud je pak volána funkce `create_server_connection`, a zároveň existuje již vytvořené TLS spojení, musí být ověřeno, že existující spojení je stále připraveno k použití a zároveň že odpovídá požadavkům klienta.

K ověření, že TLS spojení nebylo nečekaně uzavřeno protější stranou slouží nově vytvořená funkce `connection_is_still_alive`. Ta, pokud je využíváno zabezpečené spojení, nejdříve kontroluje, že od protější strany nebyla přijata zpráva typu `close_notify`. Pokud tomu tak není, nebo pokud bylo vytvořeno jen nezabezpečené TCP spojení, je následně kontrolováno i TCP spojení pomocí původní funkce `socket_is_still_alive`. Poté jsou porovnány parametry existujícího spojení s parametry, které požadoval klient v požadavku typu `CONNECT`. Tato kontrola byla doplněna i o porovnání typu spojení, tedy jestli se jedná o zabezpečené TLS spojení. Taková kontrola není sice nezbytná pro opakované použít TLS spojení, protože by nemělo dojít ke změně požadovaných parametrů mezi jednotlivými požadavky. Nezbytná je ovšem pro nezabezpečená spojení i pokud jsou TLS session sdíleny mezi jednotlivými vlákny.

Jestliže tedy parametry spojení odpovídají a spojení nebylo neočekávaně uzavřeno protější stranou, může být proces navazování nového spojení s cílovým serverem vynechán. Pokud by parametry spojení neodpovídaly požadav-

kům, případně pokud bylo spojení neočekávaně ukončeno, proxy server zašle vlastní `close_notify` zprávu. Místo původního uzavřeného spojení následně vytvoří nové, které může být použito pro vyřízení požadavku klienta.

4.6.5 Zpracování požadavků na konfigurační rozhraní

Ačkoliv byl proxy server již přizpůsoben pro zpracování HTTPS požadavků na konfigurační rozhraní (sekce 4.5.1), opakované využívání TLS spojení ovlivňuje i zpracování těchto požadavků. Při opakovaném použití TLS spojení totiž klient nezasílá požadavek typu `CONNECT`, který se původně u dotazů na konfigurační rozhraní přes zabezpečené spojení vždy předpokládal. Proto pokud je opakovaně využíváno TLS spojení s klientem, přes které klient žádal spojit s konfigurační url adresou, proxy server generuje příslušnou webovou stránku již po přijetí prvního požadavku.

4.6.6 Spolupráce s nadřazeným proxy serverem

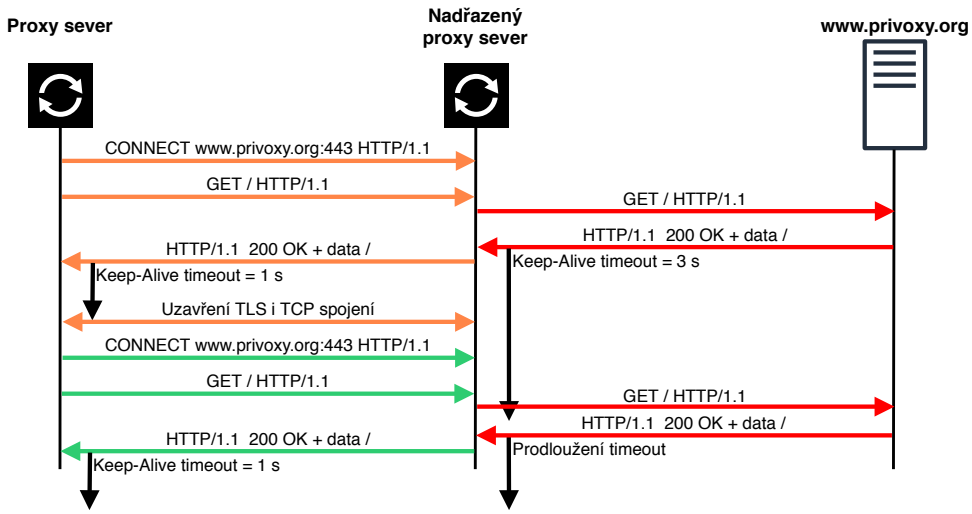
Díky použitému principu pro opakované využívání TLS sessions, který neukončuje TLS spojení po vyřízení požadavku, iniciuje proxy server uzavření TLS spojení s klientem až po vypršení časového limitu `Keep-Alive`. Tento časový limit definuje dobu, po kterou má být udržováno vytvořené TCP spojení. (Spojení s webovým serverem uzavírá proxy server ve stejný okamžik, pokud nebyla s webovým serverem vyjednána jiná hodnota parametru `Keep-Alive`, případně pokud není časový limit definován v konfiguračním souboru proxy serveru.) TLS spojení jsou tedy uzavírána těsně před uzavřením TCP spojení. Z toho vyplývají určitá specifika při použití nadřazeného proxy serveru, která jsou dána rozdílnými časovými limity `Keep-Alive` na obou proxy serverech. Obecně mohou nastat dva případy.

1. Nadřazený proxy sever má delší časový limit `Keep-Alive`

   Jak můžeme vidět na obrázku 4.7, pokud je časový limit pro uzavření spojení delší na nadřazeném proxy serveru, nedochází k žádným komplikacím. První proxy server po vypršení časového limitu uzavře spojení s nadřazeným proxy serverem, a když obdrží další požadavek na totožný server, opět jej vytvoří. Nadřazený proxy server existující spojení s webovým serverem uchovává, a jelikož nový požadavek přijde v časovém limitu, nemusí nadřazený proxy server spojení obnovovat, ale okamžitě využije již existující.

2. Nadřazený proxy server má kratší časový limit `Keep-Alive`

   Pokud je časový limit pro uzavření spojení na nadřazeném proxy serveru kratší než na předcházejícím, dochází ke zkomplikování celého procesu. Na obrázku 4.8 je zobrazeno, jak musí proxy servery reagovat. Oba proxy servery po prvním požadavku vytvoří TCP i TLS spojení,

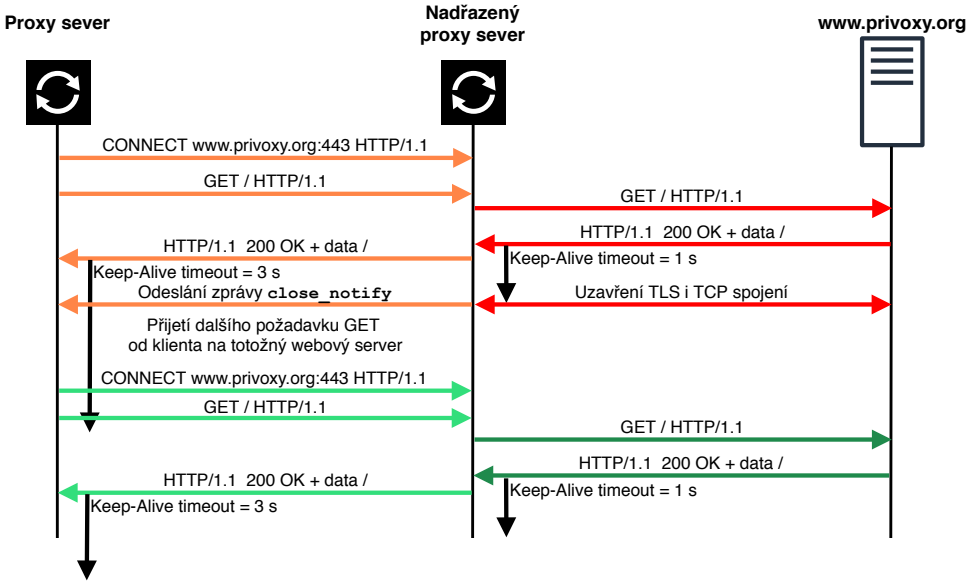


Obrázek 4.7: Opětovné použití TLS sessions s nadřazeným proxy serverem, kdy nadřazený proxy server má delší časový limit

která po vyřízení požadavku uchovávají až do uplynutí časového limitu. Nadřazený proxy server následně uzavře spojení nejen s cílovým serverem, ale i s předchozím proxy serverem. To znamená, že předcházejícímu proxy serveru zašle `close_notify` zprávu. Jelikož předcházející proxy server naslouchá požadavkům klienta, nedetekuje uzavření tohoto spojení a zjistí ho až ve chvíli, kdy přijde od klienta další požadavek typu `GET` na totožný server. V tu chvíli zjistí, že spojení bylo nadřazeným proxy serverem uzavřeno a musí být znovu vytvořeno pomocí požadavku `CONNECT`. Zbytek komunikace již probíhá standardním způsobem.

Problém ve druhém případě spočívá v tom, že první z proxy serverů uchovává aktivní TLS spojení s klientem i ve chvíli, kdy nadřazený proxy sever již uzavřel spojení s webovým serverem i předchozím proxy serverem. První z proxy serverů tak přijme od klienta požadavek typu `GET`, ale protože bylo následující spojení uzavřeno, musí jej obnovit. K tomu je nezbytné zaslat požadavek typu `CONNECT` na nadřazený proxy server. Tento požadavek ovšem není v tu chvíli od klienta k dispozici. V rámci diplomové práce byl tento problém řešen uložením celého `CONNECT` požadavku ze začátku komunikace s klientem, ovšem až po aplikování případných filtrů. Pokud tedy musí být spojení s nadřazeným proxy serverem obnoveno, je k tomu použit tento uložený požadavek. Poté proxy server pokračuje stejným způsobem jako po vytvoření nového TLS spojení.

Pokud není použit nadřazený proxy server, k této komplikaci nedochází,



Obrázek 4.8: Opětovné použití TLS sessions s nadřazeným proxy serverem, kdy nadřazený proxy server má kratší časový limit

jelikož proxy server po vypršení časového limitu odesílá `close_notify` zprávu přímo klientovi. Ten tak před odesláním dalšího požadavku zjistí, že spojení bylo uzavřeno a naváže s proxy serverem nové.

4.6.7 Přizpůsobení knihovně Mbed TLS

Součástí diplomové práce je i podpora pro využití nových rozšíření i s původní kryptografickou knihovnou Mbed TLS. Proto byly veškeré funkce potřebné pro opakované využívání TLS sessions implementovány i s použitím této knihovny. Uživatel tak při kompilaci s knihovnou Mbed TLS může toto rozšíření využít. Jelikož s touto skutečností bylo počítáno již během návrhu úprav, bylo vše navrženo tak, aby bylo v budoucnu možné jednoduše nahradit původní kryptografické knihovny jakoukoli jinou. Stačí implementovat s využitím konkrétní knihovny funkce z tabulky 4.3 tak, aby splňovaly velmi obecné rozhraní.

4.7 Sdílení TLS sessions

Aby bylo dosaženo ještě většího zrychlení během vyřizování klientových požadavků, implementuje diplomová práce sdílení TLS spojení mezi vlákny. Tato vlákna obsluhují jednotlivé klienty, a tak vytváří i požadovaný typ spojení

Tabulka 4.3: Funkce nezbytné pro začlenění nové kryptografické knihovny

Soubor	Název funkce
ssl_libressl.c ssl_mbedtls.c	[client|server]_use_ssl
	is_ssl_pending
	tunnel_established_successfully
	ssl_[send|recv]_data
	ssl_send_data_delayed
	ssl_flush_socket
	ssl_send_certificate_error
	create_[client|server]_ssl_connection
	close_client_and_server_ssl_connections
	close_[client|server]_ssl_connection
	close_ssl_connection
	was_connection_closed_by_peer
	is_server_certificate_valid

s cílovými webovými servery. Spojení mezi sebou mohou následně sdílet. Původní verze proxy serveru Privoxy již umožňuje sdílení TCP spojení mezi vlákny. Toto původní řešení bylo v rámci diplomové práce použito jako základ pro sdílení existujících TLS spojení.

4.7.1 Původní sdílení TCP spojení

Pro sdílení existujících TCP spojení využívá původní verze Privoxy pole fixní velikosti 100 spojení, které je sdílené mezi všemi vlákny proxy serveru. Do něj jsou vkládány kromě TCP soketu i parametry popisující toto spojení (např. doménové jméno, port, informace o nadřazeném proxy serveru, ...) a parametry pro spravování tohoto spojení. Mezi ně mimo jiné patří časový limit `Keep-Alive` a čas, kdy bylo spojení naposledy aktivně užíváno. Posledním důležitý parametr udává, jestli je konkrétní spojení aktuálně využíváno některým z vláken. Přístup do tohoto pole je řízen pomocí mutexů. Spojení jsou do tohoto pole vkládána ve dvou případech:

1. Nově přijatý požadavek je určený pro jiný webový server než předchozí

   Při použití nezabezpečeného spojení pro HTTP komunikaci zasílá klient v každém dotazu konkrétní doménové jméno. Požadavek tedy může vypadat například takto:

   ```
   GET http://www.privoxy.org/ HTTP/1.1
   ```

   Díky tomu může být každý z příchozích požadavků určený pro jiný webový server, aniž by bylo spojení mezi klientem a webovým serverem

uzavřeno. Pokud je tedy nový požadavek určen pro jiný webový server než předchozí, proxy server původní spojení se serverem neuzavře, ale vloží ho do zmiňovaného pole sdílených spojení. Pro nový požadavek pak vytvoří i nové TCP spojení. Jelikož při použití TLS spojení obsahuje doménové jméno pouze první požadavek typu `CONNECT`, nemůže být z nově příchozího požadavku zjištěna změna cílového serveru, a tak ani nemůže být existující TLS spojení vloženo mezi sdílená.

2. Spojení s klientem je uzavřeno dříve, než vyprší časový limit `Keep-Alive` spojení s webovým serverem

   Po vyřízení klientova požadavku vlákno zachovává spojení otevřené a čeká na další požadavek. Pokud jej klient nezašle ve stanoveném časovém limitu `Keep-Alive`, spojení s klientem je uzavřeno. Uzavření spojení může iniciovat i klient, pokud ví, že již nebude chtít zasílat další požadavky. Pokud je spojení s webovým serverem i poté aktivní a neuplynul jeho stanovený časový limit `Keep-Alive`, může být uloženo pro další použití. Výchozí `Keep-Alive` limit pro webový server může správce proxy serveru nastavit v konfiguračním souboru.

Když následně jiné vlákno přijme požadavek na spojení s určitým webovým serverem, může nejdříve zkontrolovat, jestli vhodné spojení již není vytvořené ve sdíleném poli. Pokud takové spojení existuje, není využíváno jiným vláknem, je stále aktivní a nevypršel jeho `Keep-Alive` časový limit, nastaví u něj vlákno příznak, že je již využíváno některým z aktivních vláken, a následně ho využije pro komunikaci s webovým serverem. Nemusí tak vytvářet nové spojení, jehož navazování by zpomalilo celý proces vyřizování klientova požadavku. Pokud vlákno požadavek vyřídí a spojení s klientem je uzavřeno, opět nastaví příznak v poli sdílených spojení, že je k dispozici pro ostatní vlákna. Pokud je spojení v průběhu komunikace uzavřeno, odstraní ho příslušné vlákno ze seznamu sdílených spojení. Případně jsou spojení, u kterých vypršel časový limit, uzavřena v průběhu prohledávání sdíleného pole.

4.7.2 Správa TLS parametrů sdílených spojení

Aby mohla být sdílená spojení využívána i pokud jsou zabezpečená pomocí protokolu TLS, je nezbytné sdílet kromě původního soketu i další parametry nezbytné pro použití TLS vrstvy. Pro sdílení těchto parametrů by mohl být vytvořen samostatný mechanismus. V rámci diplomové práce ale bylo zvoleno rozšíření původního postupu. Proto byly rozšířeny tři hlavní funkce pro správu sdílených spojení:

- `remember_connection`

  Tato funkce vkládá nová spojení do sdíleného pole. Aby mohly být uloženy i parametry týkající se TLS spojení, byly přidány do struktury `reusable_connection`, která je v parametru této funkce. Funkce

tyto parametry zkopíruje do struktury uložené ve sdíleném poli. Pokud ve sdíleném poli nejsou žádná volná místa, předané spojení je uzavřeno, a to včetně vrstvy TLS, pokud se jedná o zabezpečené spojení.

Nově byla funkce rozšířena i o návratovou hodnotu. Podle ní volající zjistí, jestli se podařilo spojení úspěšně uložit, případně jestli bylo funkcí uzavřeno. Na základě návratové hodnoty tedy mohou být nastaveny parametry o stavu ukládaného spojení.

- `close_unusable_connections`

  Spojení, která byla uzavřena webovým serverem, nebo kterým vypršel časový limit `Keep-Alive` jsou spravována touto funkcí. Ta prochází pole sdílených spojení, a pokud již nemohou být použita, provede jejich korektní uzavření. Proto byla původní funkce pro kontrolu, jestli je TCP spojení stále aktivní, nahrazena funkcí kontrolující i stav TLS spojení. Zároveň byla původní funkce uzavírající TCP spojení doplněna funkcí pro uzavření TLS spojení.

- `get_reusable_connection`

  Funkce prochází pole sdílených spojení a na základě předaných parametrů v něm hledá odpovídající spojení. Aby mohla funkce v případě shody vrátit i nezbytné parametry pro TLS komunikaci, byl mezi její parametry přidán ukazatel na strukturu `ssl_connection_attr`.

Díky zmíněným úpravám funkcí pro správu sdílených spojení již nejsou nutné rozsáhlejší úpravy algoritmu pro obsluhu klientových požadavků. Algoritmus byl doplněn o nastavení příznaků týkajících se opětovně použitého TLS spojení na základě návratových hodnot funkcí spravujících sdílená spojení. Přidány byly také obecné funkce pro uzavření samostatného TLS spojení a uvolnění paměti, které využívají struktury obsahující parametry tohoto spojení. Původní ekvivalenty těchto funkcí byly přizpůsobeny vždy spojení s webovým serverem nebo s klientem.

4.8 Konfigurace šifrových sad

Privoxy využívá několik typů konfiguračních souborů. Pro přiřazení akcí k jednotlivým url adresám či webovým serverů slouží soubory s příponou `.action`. Aby mohl uživatel definovat vlastní šifrovou sadu pro libovolný webový server, byla vytvořena nová akce s klíčovým slovem `set-cipher-list`. Parametrem tohoto klíčového slova je řetězec, jehož struktura se liší v závislosti na použité kryptografické knihovně. Na dalších řádkách za klíčovým slovem je nezbytné uvést seznam url adres webových serverů. Pro tyto servery bude zadaná šifrová sada využívána.

- Knihovna LibreSSL

  Uživatel může pomocí klíčových slov[11] a speciálních symbolů definovat libovolnou podmnožinu podporovaných algoritmů. Využito může být klíčové slovo `DEFAULT`, které představuje výchozí šifrovou sadu. Tuto sadu lze následně upravovat. Ukázkovou hodnotou může být například:

  ```
  DEFAULT:!RC4:!MD5
  ```

- Knihovna Mbed TLS

  Řetězec obsahuje seznam klíčových slov[12] definujících jednotlivé šifrové sady. Klíčová slova jsou oddělena znakem „:“. Každá z uvedených sad bude proxy serverem povolena. Ukázkovou hodnotou může být například:

  ```
  TLS-RSA-WITH-AES-128-CBC-SHA:...
  ```

Pokud je zadaný řetězech chybný, dotčená spojení nejsou navázána a uživatel je upozorněn v logovacím souboru. U knihovny LibreSSL je zadaný řetězec předán přímo knihovní funkci. U knihovny Mbed TLS je řetězec rozdělen na jednotlivá klíčová slova, která jsou převedena na pole odpovídajících číselných hodnot. To je dále zpracováno knihovní funkcí.

[11] Kompletní přehled podporovaných klíčových slov a syntaxe konfiguračního řetězce je k dispozici na `https://man.openbsd.org/SSL_set_cipher_list.3`

[12] Kompletní přehled podporovaných klíčových slov je k dispozici na `https://tls.mbed.org/supported-ssl-ciphersuites`

Kapitola 5

Testování dosažených výsledků

Implementované změny jsou v této kapitole testovány z několika různých pohledů. Zároveň jsou zde zhodnoceny dosažené výsledky a naznačena i vhodná budoucí rozšíření, která by navázala na tuto diplomovou práci.

5.1 Funkčnost

Nově implementovaná rozšíření bakalářské práce úspěšně umožňují filtrovat veškerou přenášenou HTTPS komunikaci v obou směrech. Tedy nejen data přijatá od webového serveru jako předchozí bakalářská práce, ale i data přijatá od klienta. Filtrovat lze přenášená data včetně HTTP hlaviček, a to i při použití nadřazeného proxy serveru. K dispozici jsou tak uživatelům veškeré funkcionality pro filtrování, které jsou obsaženy v původním Privoxy verze 3.0.28, a to nejen pro protokol HTTP, ale i pro protokol HTTPS.

Součástí provedených testů funkčnosti je i ověření podpory běžných i neobvyklých algoritmů, protokolů a nastavení zabezpečených spojení. Pro ověření funkčnosti byly použity testy z webových stránek `https://www.badssl.com`, jejichž výsledky jsou zobrazeny v tabulce 5.1. Z tabulky vyplývá, že proxy server Privoxy podporuje veškeré testované funkcionality jako moderní webové prohlížeče.

5.1.1 Kryptografické knihovny

Díky použití nové kryptografické knihovny je možné využívat Privoxy se všemi nejrozšířenějšími moderními webovými prohlížeči. Po nainstalování vlastní CA tyto prohlížeče uživatele nijak neupozorňují na bezpečnostní rizika. Byl tak odstraněn jeden ze zásadních nedostatků původní bakalářské práce. Zároveň však byla zachována kompatibilita s původně použitou kryptografickou knihovnou Mbed TLS, včetně jejích výhod a nedostatků. Uživatel si tak může sám zvolit kryptografickou knihovnu, která bude při kompilaci použita. Zároveň je

Tabulka 5.1: Srovnání Privoxy a webových prohlížečů v podpoře vybraných kryptografických protokolů a algoritmů. (Ano – zpracuje, Ne – nezpracuje)

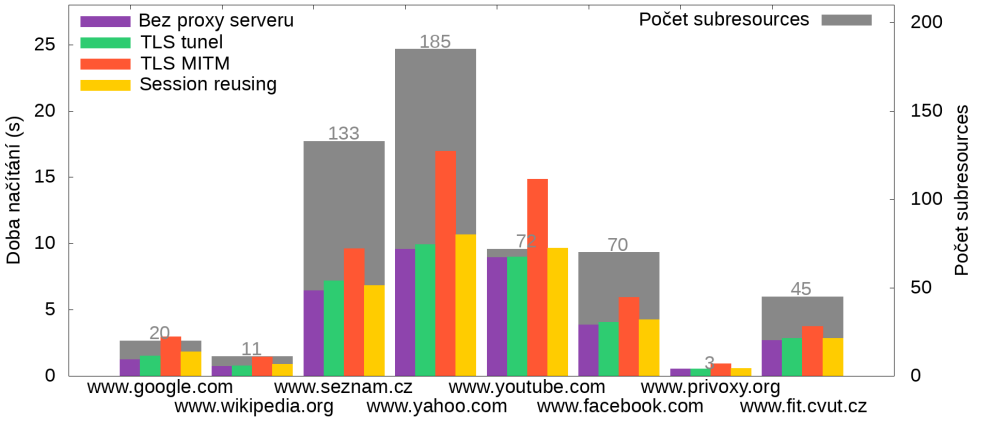
Test	Privoxy LibreSSL[13]	Privoxy MbedTLS[14]	Google Chrome[15]	Firefox[16]	Safari[17]
Neobvyklé					
1000-sans	Ano	Ne	Ano	Ano	Ano
10000-sans	Ne	Ne	Ne	Ne	Ne
sha384	Ano	Ano	Ano	Ano	Ano
sha512	Ano	Ano	Ano	Ano	Ano
rsa8192	Ano	Ano	Ano	Ano	Ano
no-subject	Ano	Ano	Ano	Ano	Ano
no-common-name	Ano	Ano	Ano	Ano	Ano
incomplete-chain	Ano	Ano	Ano	Ano	Ano
Běžné					
tls-v1-2	Ano	Ano	Ano	Ano	Ano
sha256	Ano	Ano	Ano	Ano	Ano
rsa2048	Ano	Ano	Ano	Ano	Ano
ecc256	Ano	Ano	Ano	Ano	Ano
ecc384	Ano	Ano	Ano	Ano	Ano
extended-validation	Ano	Ano	Ano	Ano	Ano
mozilla-modern	Ano	Ano	Ano	Ano	Ano
Upgrade					
hsts	Ano	Ano	Ano	Ano	Ano
upgrade	Ano	Ano	Ano	Ano	Ano
preloaded-hsts	Ano	Ano	Ano	Ano	Ano
https-everywhere	Ne	Ne	Ne	Ne	Ne

uživateli umožněno provést kompilaci bez jakékoli kryptografické knihovny, tedy se stejnými funkcionalitami jako původní Privoxy verze 3.0.28.

5.2 Výkonnost

Při implementaci nových funkcionalit pro Privoxy byl kladen důraz i na výkonnost proxy serveru. Implementováno proto bylo rozšíření umožňující opakovaně využívat vytvořená TLS spojení s klienty i s webovými servery. Díky tomu nemusí proxy server pro každé přijetí a vyřízení dotazu navazovat nová TLS spojení, čímž je značně urychleno vyřizování řady dotazů na jeden konkrétní webový server. Zrychlení je tedy patrné zejména u webových stránek obsahujících vysoký počet subresource ze společného webového serveru. Porovnání doby potřebné pro načtení vybraných webových stránek při využití různých metod implementovaných v jednotlivých verzích proxy serveru Pri-

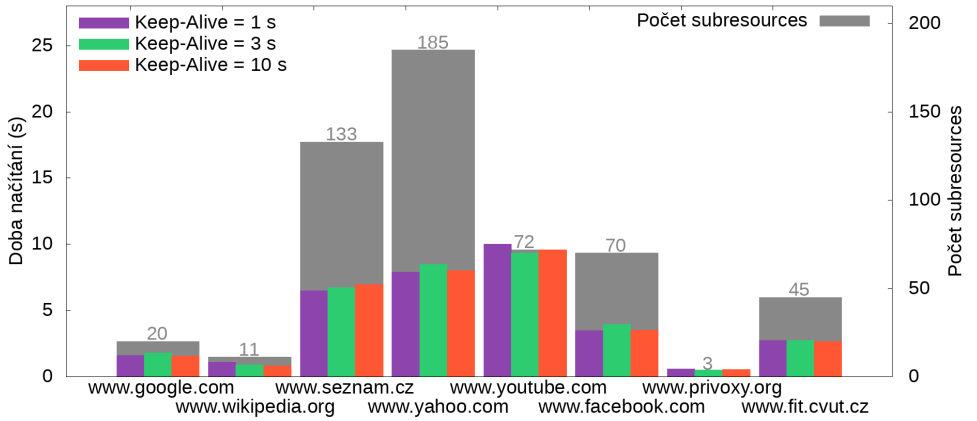
[13] LibreSSL verze 3.0.2
[14] Mbed TLS verze 2.16.6
[15] Google Chrome verze 81.0.4044.113 (64 bitů)
[16] Firefox verze 75.0 (64 bitů)
[17] Safari iPadOS 13.4.1



Obrázek 5.1: Srovnání doby načítání vybraných webových stránek podle použité metody a počtu jejích subresources. Použitý proxy server pro metody TLS tunel a TLS MITM je výstupem předchozí bakalářské práce.

voxy, případně bez použití proxy serveru, je zobrazeno v grafu na obrázku 5.1. V grafu je rovněž u každé webové stránky zobrazen počet subresources, které byly načítány jako její součást. Pro metodu TLS tunelu a TLS MITM byl použit proxy server z předchozí bakalářské práce. Aby mohlo být měření provedeno s využitím moderního webového prohlížeče, byly předem připraveny certifikáty pro jednotlivé webové stránky. Certifikáty vytvořené původní knihovnou Mbed TLS by totiž byly webovým prohlížečem označeny jako nedůvěryhodné a test by nebylo možné uskutečnit.

Jak můžeme vidět na grafu 5.1 srovnávajícím dobu potřebnou k načtení kompletní webové stránky, rozšíření proxy serveru Privoxy o opakované využívání spojení s klientem i cílovým serverem značně zrychlilo načítání webových stránek oproti metodě MITM použité v předchozí bakalářské práci. Největší zrychlení je podle očekávání u webových stránek s velkým počtem subresources. V průměru bylo na testované množině webových stránek dosaženo zrychlení 33,5 % proti původní metodě MITM. Z grafu je rovněž patrné, že pomocí sdílení TLS spojení bylo dosaženo téměř shodné doby pro odbavení požadavku klienta jako při použití metody TLS tunelu. Ta ovšem neumožňuje filtrovat HTTPS komunikaci, a ani proxy server při ní nevytváří TLS spojení. Jelikož pro měření doby načtení u metod TLS MITM a TLS tunelu byla použita implementace z předchozí bakalářské práce, může být výsledek částečně ovlivněn i rozdílnými kryptografickými knihovnami, případně dalšími změnami provedenými v rámci diplomové práce.



Obrázek 5.2: Vliv hodnoty parametru `keep-alive-timeout` na dobu načítání vybraných webových stránek

5.2.1 Vliv parametru `keep-alive-timeout` na výkonnost

Graf 5.2 zobrazuje dobu načítání webových stránek v závislosti na nastavení hodnoty parametru `keep-alive-timeout` v konfiguračním souboru proxy serveru. Ten ovlivňuje dobu, po kterou jsou spojení s klientem a webovým serverem po ukončení aktivní komunikace zachována. Z grafu vyplývá, že při jednorázovém načtení konkrétní webové stránky hodnota tohoto parametru prakticky nehraje roli (pokud tedy není nulová). Důvodem je velmi malý interval mezi příjmem jednotlivých dozatů od webového prohlížeče. Hodnota parametru by se naopak projevila při dlouhodobém prohlížení konkrétního webu, kdy jsou intervaly mezi požadavky ovlivněny i interakcí uživatele.

5.2.2 Sdílení TLS session

Sdílení TLS spojení mezi jednotlivými vlákny proxy severu nemá prakticky žádné dopady na výkon při zpracování samostatných dotazů na jednotlivé webové stránky. Výhodu přináší především při paralelním obsluhování několika klientů. Pro větší využití této funkcionality je vhodné v konfiguračním souboru `config` nastavit parametr `default-server-timeout` na hodnotu vyšší než u parametru `keep-alive-timeout`. Díky tomu roste pravděpodobnost, že spojení s webovým serverem bude udržováno i po uzavření spojení s klientem, a bude tak nabídnuto dalším vláknům. I tato implementovaná funkcionalita tak přináší další možnost, jak zvýšit výkonnost proxy severu. Vzhledem ke dříve zmiňovaným bezpečnostním rizikům v sekci 3.3.7 je konkrétní nastavení na zvážení správce proxy serveru.

5.3 Bezpečnost

Během implementace rozšíření pro proxy server Privoxy byl kladen důraz na zachování bezpečnosti. Jelikož proxy server provádí TLS handshake s cílovým serverem, provádí rovněž kontrolu jeho certifikátu a všech dalších parametrů spojení. Z tohoto důvodu byla rozšířena konfigurace proxy severu o novou akci umožňující konfigurovat seznam povolených šifrových sad pro komunikaci s webovým serverem. Díky tomu může klient nastavit úroveň zabezpečení pro libovolnou url adresu. Navíc i přesto, že byla nová rozhraní implementována s využitím nové kryptografické knihovny, zůstalo zachováno informování klienta o neúspěšné validaci certifikátu přijatého od cílového serveru, a to včetně možnosti stažení kompletního řetězce certifikátu.

Výsledná úroveň zabezpečení byla ověřena pomocí testů dostupných na `https://www.badssl.com`. Výsledky těchto testů jsou zobrazeny v tabulce 5.2 společně s výsledky moderních webových prohlížečů. V tabulce jsou zobrazeny výsledky pro výchozí nastavení bez dalších zásahů správce proxy serveru. Z tohoto důvodu jsou zejména nedostatky v části „Šifrové sady“ a „Výměna klíče“ zanedbatelné, neboť je lze odstranit vhodnou konfigurací. Ze zbylých rizik, která moderní webové prohlížeče na rozdíl od proxy serveru odhalí, je největším nedostatkem zejména chybějící kontrola revokovaných certifikátů.

5.4 Další možnosti rozvoje

Aby byl proxy server i v budoucnu konkurenceschopný, je nezbytné, aby i nadále přinášel uživatelům nové funkcionality a rozšíření. Ačkoli tato diplomová práce implementuje řadu rozšíření, nabízí se mnoho dalších možností, jak tato rozšíření nadále rozvíjet či doplňovat. Díky některým z nich se rozšíří funkcionality proxy serveru, jiné zase zvýší uživatelský komfort či zlepší zabezpečení přenášených dat.

5.4.1 Výjimky pro nevalidní certifikáty

Proxy server při filtrování HTTPS komunikace ověřuje certifikáty webových serverů a webové prohlížeče již přijmou pouze certifikát generovaný proxy serverem. V některých případech ovšem může uživatel vyžadovat připojení ke konkrétnímu webovému serveru i přes určité nedostatky certifikátu. Privoxy již umožňuje uživateli zobrazit veškeré parametry certifikátu webového serveru, pokud je označen jako nevalidní. Proxy server by ovšem kromě informací o webovém certifikátu mohl uživateli nabídnout i možnost udělení výjimky pro konkrétní certifikát. (V současnosti je umožněno udělit výjimku pouze pro určité doménové jméno, nelze specifikovat konkrétní certifikát.) Vhodnou formou pro vytvoření výjimky je například odkaz na webové stránce s informacemi o neplatném certifikátu. Po jeho otevření by proxy server při dalším zpracování tohoto certifikátu navázal spojení bez ohledu na jeho platnost.

Tabulka 5.2: Srovnání úspěšnosti Privoxy a webových prohlížečů při odhalování nedostatků SSL/TLS spojení (Ano – odhaleno, Ne – neodhaleno)

Test	Privoxy LibreSSL[18]	Privoxy MbedTLS[19]	Google Chrome[20]	Firefox[21]	Safari[22]
Validace certifikátu					
expired	Ano	Ano	Ano	Ano	Ano
wrong.host	Ano	Ano	Ano	Ano	Ano
self-signed	Ano	Ano	Ano	Ano	Ano
unrusted-root	Ano	Ano	Ano	Ano	Ano
revoked	Ne	Ne	Ano	Ano	Ano
pinning-test	Ne	Ne	Ano	Ano	Ne
no-sct	Ne	Ne	Ano	Ne	Ne
sha1-intermediate	Ne	Ano	Ano	Ano	Ano
invalid-expected-sct	Ano	Ano	Ano	Ano	Ano
Šifrové sady					
cbc	Ne	Ne	Ne	Ne	Ne
rc4-md5	Ne	Ano	Ano	Ano	Ano
rc4	Ne	Ano	Ano	Ano	Ano
3des	Ne	Ano	Ne	Ne	Ne
null	Ano	Ano	Ano	Ano	Ano
Výměna klíče					
dh480	Ano	Ano	Ano	Ano	Ano
dh512	Ano	Ano	Ano	Ano	Ano
dh1024	Ne	Ne	Ano	Ne	Ano
dh2048	Ne	Ne	Ano	Ne	Ano
dh-small-subgroup	Ne	Ne	Ano	Ne	Ano
dh-composite	Ne	Ne	Ano	Ne	Ano
Známé špatné					
Superfish	Ano	Ano	Ano	Ano	Ano
eDellRoot	Ano	Ano	Ano	Ano	Ano
DSD Test Provider	Ano	Ano	Ano	Ano	Ano
preact-cli	Ano	Ano	Ano	Ano	Ano
Webpack-dev-server	Ano	Ano	Ano	Ano	Ano
Zaniklé					
Zaniklé – sha1-2016	Ano	Ano	Ano	Ano	Ano
Zaniklé – sha1-2017	Ano	Ano	Ano	Ano	Ano

5.4.2 Nové metody pro kontrolu certifikátů

Jak vyplývá z tabulky 5.2, proxy server nekontroluje seznamy revokovaných certifikátů. Proto může klienta připojit i k webovému serveru, který se prokazuje zneplatněným certifikátem. Tento bezpečnostní nedostatek by bylo vhodné odstranit rozšířením stávající kontroly. V rámci kontroly certifikátů by mohl proxy server nabídnout i podporu CT.

[18] LibreSSL verze 3.0.2
[19] Mbed TLS verze 2.16.6
[20] Google Chrome verze 57.0.2987.133 (64 bitů)
[21] Firefox verze 75.0 (64 bitů)
[22] Safari iPadOS 13.4.1

5.4.3 Nové informativní zprávy

Proxy server již nyní informuje klienta pomocí připravených šablon o řadě problémů vzniklých během navazování spojení s webovým serverem. Jako příklad můžeme uvést zprávu typu „No such domain“, pokud se nepodaří nalézt webový server pro zadanou url adresu. S podporou zabezpečených TLS spojení ovšem přichází řada dalších specifičtějších chyb. Jedním z vhodných rozšíření je zavedení větší granularity při zasílání chybových zpráv, případně doplnění chybových stránek i pro chyby, o kterých dosud není klient informován vůbec.

5.4.4 Načítání konfiguračních souborů

Proxy server načítá řadu konfiguračních souborů. Jedním z nich je i soubor definující akce pro vybrané url adresy. Tyto akce pak například aktivují předdefinované filtry či mění parametry navázaných spojení. Pro načítání těchto akcí je v Privoxy použita proměnná typu `long`, kde jsou jednotlivé bity této proměnné přiřazeny příslušné akci. Vzhledem k omezené délce proměnné typu `long` je nyní počet akcí zcela vyčerpán a nové akce již nelze přidávat. Z tohoto důvodu by bylo vhodné upravit načítání akcí z konfiguračního souboru tak, aby nebyl jejich počet nijak limitován.

5.4.5 Protokol HTTP/2.0

Nejnovější verze protokolu HTTP přináší řadu změn oproti předchozím verzím. Z tohoto důvodu vyžaduje podpora HTTP/2.0 rozsáhlé změny na straně proxy serveru. Vzhledem k výhodám tohoto protokolu a jeho postupnému rozšiřování v rámci webové komunikace by však bylo velmi vhodné, aby jedním z dalších rozšíření Privoxy byla i podpora tohoto protokolu.

Závěr

Cílem této diplomové práce bylo analyzovat vývoj šifrovaného HTTP v aktuálních webových prohlížečích během poslední tří let, a to včetně změn v používaných certifikátech. Součástí analýzy je i zhodnocení tohoto vývoje z pohledu problematiky filtrování HTTPS komunikace pomocí proxy serveru. Na základě této analýzy pak měl být vytvořen návrh a implementace úprav proxy serveru Privoxy, které by modernizovaly předchozí implementaci podpory protokolu TLS a zároveň umožnily aplikovat funkcionality Privoxy na veškerou přenášenou HTTPS komunikaci. Práce by rovněž měla testovat původní i modernizovanou verzi Privoxy v aktuálních webových prohlížečích a diskutovat dosažené výsledky.

Kapitola 1 popisuje nejdůležitější aktualizace nejrozšířenějších webových prohlížečů během posledních třech let. Důraz byl kladen zejména na změny týkající se protokolu HTTP a certifikátů. Kapitola 2 se zabývá dopady těchto aktualizací na certifikáty a proxy servery. Popsány jsou zde nutné změny reflektující aktualizace webových prohlížečů. V kapitole 3 jsou navržena nová rozšíření a vylepšení navazující na předchozí verze Privoxy. Navržená rozšíření mimo jiné zavádí strukturování zdrojového kódu, umožňují filtrování veškeré přenášené komunikace, zjednodušují proces sestavení zdrojových kódů, urychlují zpracování požadavků klienta či rozšiřují možnosti konfigurace. Samotná implementace navržených rozšíření je detailně popsána v kapitole 4. Kapitola 5 se věnuje testování výsledného řešení a diskutuje dosažené výsledky v porovnání s předchozími verzemi Privoxy.

Výsledný proxy server umožňuje úspěšně filtrovat veškerou přenášenou HTTPS komunikaci včetně HTTP hlaviček požadavků i odpovědí a to i při použití nadřazeného proxy serveru. Díky implementovaným změnám je na rozdíl od předchozích verzí možné využít veškeré funkcionality proxy serveru i na komunikaci s moderními webovými prohlížeči, aniž by byl uživatel varován před potencionálním útokem na spojení. Metoda opakovaného používání a sdílení vytvořených spojení navíc urychlila vyřizování požadavků klienta. Na testované množině webových stránek bylo dosaženo zrychlení 33.5 % oproti

předchozí verzi, která umožňuje pouze částečné využití funkcionalit proxy serveru. Zavedena byla podpora zabezpečeného spojení pro konfigurační rozhraní proxy serveru a rozšířeny byly i možnosti konfigurace a to jak během sestavování binárního souboru Privoxy, tak při jeho využívání. Uživatel si nově může definovat šifrovou sadu pro libovolný webový server. Zároveň si může vybrat mezi nově použitou kryptografickou knihovnu LibreSSL, původní knihovnou Mbed TLS, nebo sestavení bez jakékoli kryptografické knihovny. Veškeré nové funkcionality byly implementovány pro obě tyto knihovny. Díky dalším provedeným úpravám a zjednodušením zdrojového kódu byla odhalena a odstraněna i jedna z chyb v původním zdrojovém kódu proxy serveru.

Literatura

[1] Švec, V.: *Podpora pro filtrování SSL/TLS komunikace v Privoxy*. Bakalářská práce, České vysoké učení technické v Praze, Fakulta informačních technologií, Praha, květen 2017.

[2] Mozilla and individual contributors: *Evolution of HTTP*. [online], listopad 2019, [cit. 14.2.2020]. Dostupné z: `https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/Evolution_of_HTTP`

[3] Satrapa, P.: *Jak funguje nový protokol HTTP/2*. [online], březen 2015, [cit. 9.3.2020]. Dostupné z: `https://www.root.cz/clanky/jak-funguje-novy-protokol-http-2/`

[4] *Usage statistics of HTTP/2 for websites*. [online], únor 2020, [cit. 10.2.2020]. Dostupné z: `https://w3techs.com/technologies/details/ce-http2`

[5] Böck, J.: *Provable secure RSA Signatures and their Implementation*. [online], květen 2011, [cit. 14.2.2020]. Dostupné z: `https://rsapss.hboeck.de/rsapss.pdf`

[6] Rescorla, E.: *The Transport Layer Security (TLS) Protocol Version 1.3*. [online], srpen 2018, [cit. 14.2.2020]. Dostupné z: `https://tools.ietf.org/html/rfc8446`

[7] *Chrome Platform Status*. [online], [cit. 14.2.2020]. Dostupné z: `https://www.chromestatus.com/features`

[8] Rescorla, E.: *The Transport Layer Security (TLS) Protocol Version 1.2*. [online], srpen 2008, [cit. 28.2.2020]. Dostupné z: `https://tools.ietf.org/html/rfc5246`

[9] Stevens, M.; Bursztein, E.; aj.: *The first collision for full SHA-1*. [online], 2017, [cit. 15.2.2020]. Dostupné z: `https://eprint.iacr.org/2017/190.pdf`

[10] Krasnov, V.: *It takes two to ChaCha (Poly)*. [online], duben 2016, [cit. 29.2.2020]. Dostupné z: https://blog.cloudflare.com/it-takes-two-to-chacha-poly/

[11] Mozilla and individual contributors: *Set-Cookie*. [online], březen 2020, [cit. 12.3.2020]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Set-Cookie

[12] Rescorla, E.: *HTTP Over TLS*. [online], květen 2000, [cit. 20.2.2020]. Dostupné z: https://tools.ietf.org/html/rfc2818

[13] Makarov, L.: *Say goodbye to URLs with embedded credentials*. [online], duben 2017, [cit. 18.3.2020]. Dostupné z: https://medium.com/@lmakarov/say-goodbye-to-urls-with-embedded-credentials-b051f6c7b6a3

[14] *What is Certificate Transparency*. [online], [cit. 14.2.2020]. Dostupné z: https://www.certificate-transparency.org/what-is-ct

[15] Laurie, B.; Langley, A.; Kasper, E.: *Certificate Transparency*. [online], červen 2013, [cit. 22.2.2020]. Dostupné z: https://tools.ietf.org/html/rfc6962

[16] Sleevi, R.: *Certificate Transparency Policy*. [online], duben 2017, [cit. 16.3.2020]. Dostupné z: https://groups.google.com/a/chromium.org/forum/#!msg/ct-policy/sz_3W_xKBNY/6jq2ghJXBAAJ

[17] Mozilla and individual contributors: *Using the Notifications API*. [online], únor 2020, [cit. 24.2.2020]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/Notifications_API/Using_the_Notifications_API

[18] McGrew, D. A.: *An Interface and Algorithms for Authenticated Encryption*. [online], leden 2008, [cit. 29.2.2020]. Dostupné z: https://tools.ietf.org/html/rfc5116

[19] Ghedini, A.; Vasiliev, V.: *Transport Layer Security (TLS) Certificate Compression draft-ietf-tls-certificate-compression-03*. [online], duben 2018, [cit. 2.3.2020]. Dostupné z: https://tools.ietf.org/html/draft-ietf-tls-certificate-compression-03

[20] Kingston, J.: *Restricting AppCache to Secure Contexts*. [online], únor 2018, [cit. 14.2.2020]. Dostupné z: https://blog.mozilla.org/security/2018/02/12/restricting-appcache-secure-contexts/

[21] Fielding, R. T.; Reschke, J. F.: *Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content*. [online], červen 2014, [cit. 21.2.2020]. Dostupné z: https://tools.ietf.org/html/rfc7231

[22] Popov, A.; Nystroem, M.; Balfanz, D.; aj.: *The Token Binding Protocol Version 1.0*. [online], říjen 2018, [cit. 2.3.2020]. Dostupné z: https://tools.ietf.org/html/rfc8471

[23] Popov, A.; Nystroem, M.; Balfanz, D.; aj.: *Token Binding over HTTP*. [online], říjen 2018, [cit. 3.3.2020]. Dostupné z: https://tools.ietf.org/html/rfc8473

[24] Harper, N.: *Intent to Remove: Token Binding*. [online], [cit. 24.4.2020]. Dostupné z: https://groups.google.com/a/chromium.org/forum/#!topic/blink-dev/OkdLUyYmY1E/discussion

[25] O'Brien, D.; Sleevi, R.; Whalley, A.; aj.: *Chrome's Plan to Distrust Symantec Certificates*. [online], září 2017, [cit. 15.3.2020]. Dostupné z: https://security.googleblog.com/2017/09/chromes-plan-to-distrust-symantec.html

[26] Evans, C.; Palmer, C.; Sleevi, R.: *Public Key Pinning Extension for HTTP*. [online], duben 2015, [cit. 3.3.2020]. Dostupné z: https://tools.ietf.org/html/rfc7469

[27] Yasskin, J.: *Web Packaging draft-yasskin-dispatch-web-packaging-00*. [online], červen 2017, [cit. 4.3.2020]. Dostupné z: https://tools.ietf.org/id/draft-yasskin-dispatch-web-packaging-00.html

[28] Yasskin, J.: *Mixed Content*. [online], srpen 2016, [cit. 6.3.2020]. Dostupné z: https://www.w3.org/TR/mixed-content/

[29] Merewood, R.: *SameSite cookies explained*. [online], květen 2019, [cit. 6.3.2020]. Dostupné z: https://web.dev/samesite-cookies-explained/

[30] Dierks, T.; Rescorla, E.: *The Transport Layer Security (TLS) Protocol Version 1.1*. [online], duben 2006, [cit. 8.3.2020]. Dostupné z: https://tools.ietf.org/html/rfc4346

[31] Thomson, M.: *Removing Old Versions of TLS*. [online], říjen 2018, [cit. 7.3.2020]. Dostupné z: https://blog.mozilla.org/security/2018/10/15/removing-old-versions-of-tls/

[32] Smotrankov, A.: *How does TLS 1.3 protect against downgrade attacks?* [online], [cit. 9.3.2020]. Dostupné z: https://blog.gypsyengineer.com/en/security/how-does-tls-1-3-protect-against-downgrade-attacks.html

[33] Barnes, R.: *Deprecating Non-Secure HTTP*. [online], duben 2015, [cit. 11.3.2020]. Dostupné z: https://blog.mozilla.org/security/2015/04/30/deprecating-non-secure-http/

[34] Mozilla Developer Network and individual contributors: *PKI:CT*. [online], prosinec 2014, [cit. 15.3.2020]. Dostupné z: https://wiki.mozilla.org/PKI:CT

[35] MDN contributors: *Certificate Transparency*. [online], července 2019, [cit. 17.3.2020]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/Security/Certificate_Transparency

[36] Privoxy: *Privoxy Frequently Asked Questions*. [online], [cit. 21.3.2020]. Dostupné z: https://www.privoxy.org/faq/misc.html

[37] Morton, B.: *What's the Difference Between a Public and Private Trust Certificate?* [online], březen 2019, [cit. 22.3.2020]. Dostupné z: https://www.entrustdatacard.com/blog/2019/march/difference-between-a-public-and-private-trust-certificate

[38] Campbell, B.: *HTTPS Token Binding with TLS Terminating Reverse Proxies*. [online], červen 2018, [cit. 23.3.2020]. Dostupné z: https://tools.ietf.org/id/draft-ietf-tokbind-ttrp-05.html

[39] MDN contributors: *HTTP Public Key Pinning (HPKP)*. [online], únor 2020, [cit. 25.3.2020]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Public_Key_Pinning

[40] Munshi, A.: *How HTTP proxies read TLS encrypted traffic from browsers ?* [online], říjen 2017, [cit. 25.3.2020]. Dostupné z: https://medium.com/@ethicalevil/how-http-proxies-read-tls-traffic-from-browsers-f15364e91226

[41] Privoxy: *Privoxy 3.0.28 User Manual*. [online], [cit. 4.2.2020]. Dostupné z: https://www.privoxy.org/user-manual/

[42] Privoxy: *Announcing Privoxy 3.0.28 stable*. [online], [cit. 4.4.2020]. Dostupné z: https://www.privoxy.org/announce.txt

[43] Stevens, R.: *Whats the difference between select() and poll()?* [online], [cit. 4.4.2020]. Dostupné z: http://www.unixguide.net/network/socketfaq/2.14.shtml

Příloha A

Seznam použitých zkratek

AEAD Authenticated Encryption with Additional Data

ALPN Application-Layer Protocol Negotiation

CA Certifikační autorita

CRL Certificate revocation list

CSRF Cross-site request forgery

CT Certificate Transparency

ECDSA The Elliptic Curve Digital Signature Algorithm

EKM Exported Keying Material

HPKP HTTP Public Key Pinning

HSTS HTTP Strict Transport Security

HTTP Hyper–Text Transfer Protocol

HTTPS Hyper–Text Transfer Protocol Secure

IP Internet Protocol

MAC Message Authentication Code

MITM Man In The Middle

NIST National Institute of Standards and Technology

NTLM NT LAN Manager

OAEP Optimal Asymmetric Encryption Padding

PDF Portable Document Format

PKI Public Key Infrastructure

PSK Pre-shared key

PSS Probabilistic Signature Scheme

RFC Request for Comments

RTT Round-trip time

SCT Signed Certificate Timestamp

SPKI Subject Public Key Information

SSL Secure Sockets Layer

TCP Transmission Control Protocol

TLS Transport Layer Security

TPM Trusted Platform Module

TTRP TLS Terminating Reverse Proxy

URI Uniform Resource Identifier

URL Uniform Resource Locator

WWW World Wide Web

XSS Cross-site scripting

Příloha B

Uživatelská příručka

Pro instalaci a spuštění proxy serveru Privoxy včetně nově implementovaných rozšíření musí uživatel připravit určité nástroje a doplňky. Tato příloha stručně popisuje postup pro instalaci Privoxy a jeho následné využití ve webovém prohlížeči.

B.1 Požadavky pro instalaci

Pro sestavení zdrojového kódu proxy serveru Privoxy včetně implementovaných rozšíření jsou potřeba nástroje autoconf, GNU make a kompilátor jazyka C. Zároveň je potřeba zvolit jednu ze dvou podporovaných kryptografických knihoven. Tedy buď knihovnu LibreSSL nebo Mbed TLS. Zkompilovaný zdrojový kód knihovny je nezbytné umístit do složek `libressl` respektive `mbedtls` v kořenové adresáři proxy serveru.

Aby mohl proxy server filtrovat HTTPS komunikaci, je nezbytné připravit i vlastní CA (včetně jejího veřejného a soukromého klíče), soubor obsahující důvěryhodné CA a adresářovou strukturu, ve které může proxy server ukládat vytvořené certifikáty. Příslušné parametry je možné nastavit v konfiguračním souboru `config`.

B.2 Instalace

Pro sestavení zdrojového kódu proxy serveru včetně implementovaných rozšíření byly upraveny příslušné konfigurační soubory. Sestavení projektu bylo úspěšně testováno s použitím nástrojů Cygwin64 Terminal, Windows Subsystem for Linux (WSL) – Ubuntu 18.04 LTS a v terminálu operačního systému Linux Mint v19.3 x86. Pokud uživatel používá jeden z těchto nástrojů, stačí pomocí následujících příkazů připravit soubor `makefile` a s jeho pomocí následně sestavit binární soubor:

```
autoheader && autoconf && ./configure
make
```

Vznikne tak binární soubor ke spuštění proxy serveru Privoxy. Konkrétní kryptografickou knihovnu lze zvolit pomocí parametrů `--disable-libressl` nebo `--enable-mbedtls` u `configure` skriptu.

Testováno bylo i sestavení zdrojového kódu pomocí nástroje MingW. Pro kompilaci knihovny LibreSSL pomocí MingW lze použít tyto příkazy:

```
CC=i686-w64-mingw32-gcc CPPFLAGS=-D__MINGW_USE_VC2005_COMPAT
./configure --host=i686-w64-mingw32
 make
```

K vygernerování `makefile` pro sestavení Privoxy pak slouží následující příkazy:

```
autoheader && autoconf
./configure --host=i686-w64-mingw32 --enable-mingw32
        --disable-pthread
```

Následně je nezbytné ve vygenerovaném souboru `GNUmakefile` doplnit definici `SSL_LIBRE_LIB` a `SSL_MBEDTLS_LIB` o přepínač „-lws2_32“. Poté již lze úspěšně použít příkaz `make` k sestavení binárního souboru.

B.3 Nastavení webového prohlížeče

Pro využití proxy serveru webovým prohlížečem je ještě nezbytné nastavit ip adresu a port na kterém proxy server naslouchá. To je možné v nastaveních webového prohlížeče, případně v nastaveních operačního systému. Příslušné parametry je rovněž možné upravit v konfiguračním souboru `config`.

B.4 Konfigurace proxy serveru

Možnosti konfigurace proxy serveru byly v této práci rozšířeny o definici vlastní šifrové sady pro vybrané webové servery. Uživatel může tuto sadu definovat pomocí klíčového slova `set-cipher-list` v souboru `user.action`. Parametr tohoto klíčového slova definuje povolenou šifrovou sadu a jeho struktura se může lišit v závislosti na použité kryptografické knihovně.

- Knihovna LibreSSL

  Povolení všech šifrových sad, které nevyužívají vybrané algoritmy:

  ```
  {+set-cipher-list{ALL:!aNULL:!eNULL:!MD5:!RC4}}
  url{www.privoxy.org}
  ```

- Knihovna Mbed TLS

  Povolení pouze jediné šifrové sady:

  ```
  {+set-cipher-list{TLS-RSA-PSK-WITH-RC4-128-SHA}}
  www.privoxy.org
  ```

Příloha C

Obsah přiloženého CD

- `readme.txt` ..... Stručný popis obsahu CD
- `builds`
  - `original_privoxy_3.0.28_win32.zip` ..... Privoxy 3.0.28
  - `implementation_libressl_win32.tar.gz` .. Implementace s LibreSSL
- `source_files`
  - `basic_configuration_files.tar.gz` ..... Konfigurační soubory
  - `patch_files.tar.gz` ..... Patch soubory
  - `git_repository_export.tar.gz` ..... Export Git repozitáře
  - `implementation.tar.gz` ..... Zdrojové kódy implementace
  - `thesis.tar.gz` ..... Zdrojová forma práce ve formátu LaTeX
  - `original_source_files`
    - `mbedtls-2.16.6-gpl.tgz` ..... Knihovna Mbed TLS
    - `libressl-3.1.1.tar.gz` ..... Knihovna LibreSSL
    - `privoxy-3.0.26-stable-src.tar.gz` ..... Privoxy 3.0.26
    - `privoxy-3.0.28-stable-src.tar.gz` ..... Privoxy 3.0.28
- `text`
  - `DP_Vaclav_Svec_2020.pdf` ..... Text práce ve formátu PDF