ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Bezpečnostní incidenty v SSL/TLS implementacích
Student:	Jan Král
Vedoucí:	Ing. Josef Kokeš
Studijní program:	Informatika
Studijní obor:	Informační technologie
Katedra:	Katedra počítačových systémů
Platnost zadání:	Do konce letního semestru 2017/18

Pokyny pro vypracování

Seznamte se s bezpečnostními incidenty týkající se protokolů SSL/TLS a jejich implementací (např. chyby Heartbleed, FREAK a podobné). Zdokumentujte jejich příčiny, omezení, důsledky a nápravu. Vyberte několik vhodných slabin a vytvořte aplikaci, která poslouží k demonstraci těchto slabin uživatelům. Diskutujte náročnost takové implementace a doporučení, jak podobným útokům v budoucnosti zabránit.

Seznam odborné literatury

Dodá vedoucí práce.

prof. Ing. Róbert Lórencz, CSc.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 17. ledna 2017

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra počítačových systémů

Bakalářská práce

Bezpečnostní incidenty v SSL/TLS implementacích

Jan Král

Vedoucí práce: Ing. Josef Kokeš

3. ledna 2018

Poděkování

Rád bych tímto poděkoval vedoucímu mé práce Ing. Josefu Kokešovi především za vstřícný přístup a bezproblémovou komunikaci při tvorbě této práce. Dále pak mým rodičům za podporu během celého studia a v neposlední řadě mé přítelkyni za trpělivost, kterou se mnou měla při dopisování 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 3. ledna 2018 .....

Odkaz na tuto práci

Král, Jan. *Bezpečnostní incidenty v SSL/TLS implementacích.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2018.

Abstrakt

Tato bakalářská práce se věnuje protokolu SSL/TLS a několika s ním souvisejícím síťovým útokům a bezpečnostním slabinám z posledních let. Jejím cílem je představit a analyzovat tyto útoky, jejich možnosti a jejich případná omezení. Konkrétně se práce věnuje útokům Heartbleed, DROWN, FREAK, Triple handshake a SWEET32. Dále je součástí práce praktická část demonstrující dvě různé varianty útoku Heartbleed. Výsledky této práce umožňují názorně se přesvědčit o reálnosti a nebezpečnosti daných útoků.

Klíčová slova protokol SSL/TLS, síťová bezpečnost, síťové útoky, Heartbleed, DROWN, FREAK, Triple handshake, SWEET32

Abstract

This bachelor's thesis is focused on the SSL/TLS protocol and some related network attacks and security weaknesses from recent years. Its purpose is to describe and analyze these attacks, their capabilities and possible limitations. The thesis is specifically focused on Heartbleed, DROWN, FREAK, Triple handshake and SWEET32 attacks. The practical part of this thesis demonstrates two different types of the Heartbleed attack. The results help readers to understand the feasibility and danger of given attacks.

Keywords protocol SSL/TLS, network security, network attacks, Heartbleed, DROWN, FREAK, Triple handshake, SWEET32

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Internet je v dnešní době již nedílnou součástí životů mnoha lidí. Můžeme jeho prostřednictvím vzájemně komunikovat, sdílet a vyhledávat informace nebo dokonce spravovat naše soukromé finance. Vzhledem k tomu, co internet umožňuje, je nesmírně důležité jeho kvalitní zabezpečení. Bezpečnost na internetu se tak stává neustále probíraným a také čím dál důležitějším tématem.

Cílem této práce je seznámit zájemce nejen s bezpečností na internetu, ale především s některými možnými bezpečnostními riziky a představit některé příklady chyb a problémů, které se v posledních letech v síťové bezpečnosti objevily.

Téma bezpečnostních incidentů v implementacích protokolu SSL/TLS jsem zvolil z toho důvodu, že se jedná o protokol, který je dnes nedílnou součástí internetu a má za cíl zajišťovat jeho bezpečnost. Umožňuje nám využívat internet i k tak citlivým činnostem jako je soukromá komunikace nebo například správa našich financí prostřednictvím internetového bankovnictví. S důležitostí tohoto protokolu se však také objevují neustále nové pokusy o překonání jeho bezpečnosti a s jeho komplexností také chyby v jeho implementacích či přímo v použitých principech.

Konkrétně se tato práce zabývá popisem a analýzou bezpečnostních chyb a útoků Heartbleed, DROWN, FREAK, Triple handshake a SWEET32. Jsou to příklady jen některých větších bezpečnostních problémů objevených v posledních letech. Součástí práce je také praktická část. Tu tvoří aplikace, které slouží k demonstraci dvou rozdílných variant útoku Heartbleed.

V první části práce je podrobně popsán samotný protokol SSL/TLS včetně jeho stručné historie. Další částí je pak popis a analýza výše zmíněných útoků, zahrnující mimo jiné vysvětlení jejich principu, možností a jejich případných omezení. Následuje praktická část práce zabývající se postupně dvěma variantami útoku Heartbleed. Nachází se zde popis, testování a posouzení výsledků aplikací demonstrujících tento útok.

Kapitola 1

Protokoly SSL/TLS

S postupným rozšiřováním internetu bylo zřejmé, že se bude také objevovat čím dál více i jeho komerčních využití. Dnes je naprosto samozřejmé využívání například internetového bankovnictví, internetových obchodů a dalších podobných služeb. Vzhledem k tomu, že hlavní myšlenkou internetu bylo a je otevřené a veřejné sdílení informací, bylo potřeba nějakým způsobem zajistit bezpečnost a utajení citlivých informací souvisejících nejen s komerčními internetovými aplikacemi. Zároveň bylo žádoucí, aby se takové bezpečnostní opatření obešlo bez zvláštních a náročných úkonů na straně koncového uživatele, tedy aby veškeré potřebné procesy byly automatickými funkcemi například internetového prohlížeče. Krátce po uvedení prvního rozšířenějšího prohlížeče Mosaic 1.0 v roce 1993 začala společnost Netscape Comunications pracovat na první verzi protokolu SSL (*Secure Sockets Layer*) a o několik let později na jeho nástupci, protokolu TLS (*Transport Security Layer*).[1]

1.1 Historie

V polovině roku 1994 byla hotová první verze protokolu SSL 1.0. Tato verze však nebyla zveřejněna, neboť měla stále mnoho chyb a nedostatků, například nezajišťovala integritu přenášených dat. Známé problémy byly rychle opraveny a ke konci roku 1994 vzniklo SSL 2.0, které už například obsahovalo ověřování dat pomocí MD5 hashů místo jednoduchých kontrolních součtů. Tato verze se již dostala na veřejnost. Nejprve jako součást prohlížeče Netscape Navigator. V roce 1995 si Netscape zažádal o patentování SSL, aby zamezil vzniku dalších podobných protokolů. Patent mu byl udělen v roce 1997, přičemž se následně společnost patentu vzdala, aby mohl bezpečnostní protokoly každý volně využívat.[1]

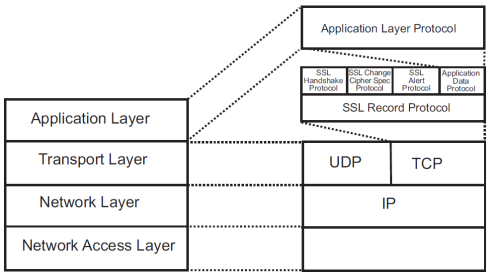
V půlce roku 1995, tedy předtím, než Netscape získal patent pro SSL, přišla společnost Microsoft se svým prohlížečem Internet Explorer a s vlastním bezpečnostním protokolem PCT (*Private Comunication Technology*). Proto-

kol Microsoftu byl technicky velmi podobný SSL 2.0 a dokonce s ním byl v některých směrech kompatibilní. PCT přineslo některá vylepšení proti SSL 2.0. Tyto změny byly zaneseny i do SSL ve verzi 3.0, která vznikla ještě před koncem roku 1995 a plně zveřejněna byla v listopadu 1996. Změny v nové verzi zahrnovaly například používání různých klíčů pro šifrování zpráv a pro jejich podepisování nebo vylepšení generování hashů pomocí algoritmu SHA-1. V reakci na SSL 3.0 přišel Microsoft s dalším protokolem, a to STLP (*Secure Transport Layer Protocol*). Pro vyřešení zmatků v oblasti bezpečnostních protokolů vznikla IETF TLS WG (*Internet Engineering Task Force Transport Layer Security Working Group*), která měla za úkol vyvinout sjednocený bezpečnostní protokol TLS.[1]

První verze TLS byla hotová ke konci roku 1997, ale kvůli různým problémům a nacházeným nedostatkům bylo TLS 1.0 zveřejněno až v lednu 1999. Oproti předchozímu protokolu SSL obsahoval nový protokol například šifrování pomocí šifry DES a 3DES nebo algoritmus Diffie-Hellman pro výměnu klíčů. Ve výsledku se však od SSL 3.0 lišil méně než SSL 3.0 od verze 2.0. Vývoj protokolu TLS dále pokračoval. V dubnu roku 2006 byla zveřejněna jeho nová verze TLS 1.1 a nakonec v srpnu 2008 TLS 1.2, která je používána dodnes. V současné době probíhá vývoj a doplňování nového TLS 1.3, které má například vyřadit již nedostačující algoritmy jako například MD5 či SHA-224 a naopak přidat některé nové bezpečnější.[1]

1.2 Popis protokolů

Protokol SSL/TLS je bezpečnostní protokol pro komunikaci mezi klientem a serverem. Zajišťuje autentizaci stran komunikace i původu dat, důvěrnost a integritu spojení. Nezaručuje však nepopiratelnost původu ani doručení a nemůže být použit například k ověřeným elektronickým podpisům. V upraveném TCP/IP modelu tvoří SSL/TLS mezivrstvu mezi vrstvou transportní a aplikační. Sám se ještě dělí na dvě podvrstvy, přičemž ta vyšší z nich se skládá ze čtyř dílčích protokolů, jak je znázorněni na obrázku 1.1. Tato kapitola popisující SSL/TLS čerpá z [1].



Obrázek 1.1: Rozšířený model TCP/IP s SSL vrstvou [1]

1.2.1 Protokol záznamové vrstvy

Tou nižší podvrstvou, která je blíže transportní vrstvě, je protokol záznamové vrstvy (*SSL/TLS Record protocol*). Ten má za úkol zajišťovat samotnou bezpečnou komunikaci, tedy zašifrovat a zabezpečit data od vyšší vrstvy v případě odesílání, nebo dešifrovat a ověřit přijímaná data před jejich předáním vyšším vrstvám.

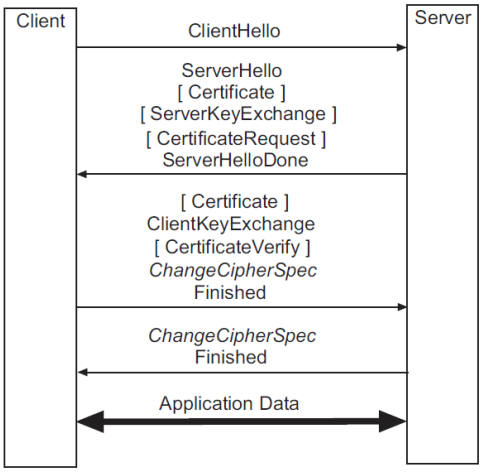
Data přijatá od vyšších protokolů a vrstev jsou nejprve fragmentována na bloky o velikosti 214 bytů či méně. Ty jsou převedeny do takzvané *SSL/TLSPlaintext* struktury. Při tomto procesu nejsou dodržována rozdělení zpráv klienta, takže jeden *SSL/TLSPlaintext* může obsahovat i více zpráv. V následující fázi dochází volitelně ke kompresi dat, pokud byla dohodnuta kompresní metoda. Umožňovat kompresi před zašifrováním je důležité, neboť zašifrovaná data již nejde komprimovat vzhledem k jejich zdánlivé náhodnosti. Dalším významem komprese dat je zvýšení entropie v posílané zprávě a další snížení šancí útočníka získat o původním textu nějaké informace. *SSL/TLSPlaintext* je tedy převeden do struktury *SSL/TLSCompressed*. Pokud není zvolena žádná metoda komprese, jsou *SSL/TLSPlaintext* a *SSL/TLSCompressed* identické. Následně je k danému bloku připojen kontrolní součet a po té je blok zašifrován, čímž vzniká struktura *SSL/TLSCiphertext*. V posledním kroku je připojena hlavička a z *SSL/TLSCiphertext* se stává kompletní SSL/TLS záznam. Hlavička obsahuje 8bitovou informaci o typu zprávy, tedy od jakého z dílčích protokolů vyšší vrstvy pochází, dále informaci o použité verzi protokolu a také o délce samotné zprávy.

1.2.2 Protokol pro navázání spojení

Protokol pro navázání spojení (*SSL/TLS Handshake Protocol*) je základní částí SSL/TLS. Umožňuje komunikujícím stranám autentizaci protějšku a dohodnutí šifrovací metody pro ochranu dat a případně i kompresní metody, které budou použity pro dané spojení. Protokol se skládá z několika typů zpráv v daném pořadí, jak je znázorněno na obrázku 1.2.

Nejprve klient pošle zprávu *ClientHello*. Ta obsahuje kromě standardní SSL/TLS hlavičky několik podstatných částí:

- 2 byty obsahující informaci o tom, jakou nejvyšší verzi protokolu klient podporuje.
- 32 bytů, které se skládají ze 4 bytů obsahujících čas a 28 bytů s náhodně vygenerovanou hodnotou, která je následně použita při šifrování, musí být tedy dostatečně náhodná a nepředpověditelná.
- 1 byte pro délku session ID, tedy identifikátoru spojení. V případě, že je tento byte nastaven na nulu, znamená to, že klient neobnovuje již navázané spojení, ale má zájem o zcela nové s novým vyjednáním jeho parametrů.



Obrázek 1.2: Průběh SSL handshake [1]

- Následuje posloupnost bytů o této délce, která případně obsahuje vlastní session ID. Jeho délka může být maximálně 32 bytů.
- 2 byty, které udávají počet šifrovacích metod podporovaných klientem.
- Následuje seznam podporovaných šifer. Pro každou šifru jsou vyhrazeny dva byty a seznam je seřazený sestupně podle preferencí klienta.
- 2 byty pro počet kompresních metod podporovaných klientem.
- Seznam kompresních metod seřazený, stejně jako v případě šifer, podle preferencí klienta.

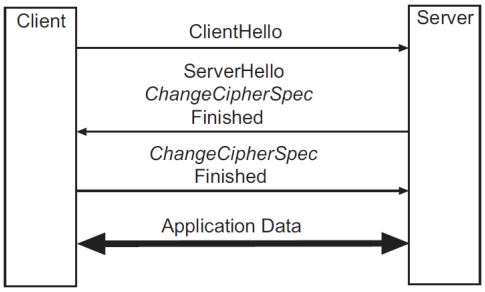
Následně server pošle zpět 2-5 zpráv. Vždy musí poslat *ServerHello* a *ServerHelloDone*. *ServerHello* je odpovědí na *ClientHello* a je vůči němu téměř identická. Jediné rozdíly jsou v označení typu zprávy a v tom, že server již nezasílá seznam šifer a kompresních metod, ale pouze od každého jednu konkrétní, která je pro dané spojení zvolena ze seznamu poskytnutého klientem. Následně může server volitelně zaslat zprávu *Certificate*, pokud se chce prokázat certifikátem a zaslat tedy veřejný klíč svého certifikátu. Dále v případě potřeby zprávu *ServerKeyExchange*, která obsahuje data potřebná pro uskutečnění výměny klíčů odpovídající dohodnutému algoritmu pro jejich výměnu (RSA, Diffie-Hellman, nebo FORTEZZA), a zprávu *CertificateRequest*, pokud server požaduje autentizaci klienta. Zpráva *ServerHelloDone* již pak slouží jen k označení konce této části navazování spojení.

V dalším kroku pošle klient 3-5 zpráv. Zprávu *Certificate*, stejně jako v předchozím kroku server, v případě, že je po něm požadována autentizace, následně zprávu *ClientKeyExchange*, jejíž obsah závisí na zvoleném algoritmu

pro výměnu klíčů. Dále zprávu *CertificateVerify*. To v případě, že klient již serveru zaslal svůj certifikát. Tato zpráva slouží k potvrzení vlastnictví příslušného certifikátu a je podepsána odpovídajícím soukromým klíčem. Po té ještě zprávu *ChangeCipherSpec*, která odpovídá protokolu pro změnu používané šifry popsanému níže, a poslední zprávu *Finished*. Ta je jako první chráněna dohodnutým šifrováním a slouží k potvrzení toho, že dohodnutí šifry a celého spojení bylo úspěšné.

Nyní ještě server zašle vlastní zprávy *ChangeCipherSpec* a *Finished*. Tím je navázaní spojení kompletní a dále probíhá výměna dat pomocí protokolu aplikační vrstvy. Existuje ještě zpráva *HelloRequest*, ale ta je málokdy používaná. Umožňuje serveru zažádat o opětovné vyjednání spojení.

V případě, že mají klient a server zájem obnovit, či duplikovat někdy již navázané spojení, může být tento proces podstatně zjednodušen a proveden pomocí menšího množství zpráv. To je znázorněno na obrázku 1.3.



Obrázek 1.3: Průběh zkráceného SSL handshake [1]

1.2.3 Protokol pro změnu používané šifry

Protokol pro změnu používané šifry (*SSL/TLS Change Cipher Spec Protocol*) umožňuje komunikujícím stranám změnit způsob šifrování posílaných dat. Na rozdíl od protokolu pro navázání spojení, při kterém se strany na metodách dohodnou, se tento protokol již stará o konkrétní nastavení a aplikování parametrů zvolené šifry.

Jedná se o jednoduchý protokol, který se skládá pouze ze zprávy *ChangeCipherSpec*. Ta je komprimována a zašifrována pomocí momentálně nastavených algoritmů. V případě navázání nového spojení obvykle není žádné šifrování ani komprimování nastaveno. Tato zpráva se skládá z výše zmíněné SSL/TLS hlavičky a jednoho bytu, který obsahuje informaci o nově zvolené šifře a metodě komprimace.

1.2.4 Protokol pro hlášení chyb

Protokol pro hlášení chyb (*SSL/TLS Alert Protocol*) dává komunikujícím stranám možnost signalizovat případné problémy a chyby odpovídajícími chybovými hláškami.

Každá zpráva tohoto protokolu obsahuje jednak informaci o závažnosti chyby a pak její popis. Byte pro informaci o závažnosti může mít hodnotu 1 nebo 2. Úroveň 1 značí varování a záleží na příjemci, jak na takovou chybu zareaguje. Úroveň 2 označuje fatální chybu a její přijetí musí mít za následek okamžité přerušení spojení. V případě chyb, které nemají předdefinovanou úroveň závažnosti, záleží na odesílateli zprávy, jakou hodnotu jim přidělí. Pro popis chyby je určen pouze jeden byte, přičemž jeho hodnota se rovná kódu dané chyby. Hodnoty kódů a příslušné chyby naleznete v tabulce 1.1.

Tabulka 1.1: Chybové kódy SSL/TLS [1]

chyba	kód	popis
close_notify	0	Odesílatel upozorňuje, že již nepošle další zprávy.
unexpected_message	10	Obdržení nečekané zprávy.
bad_record_mac	20	Obdržená zpráva obsahuje špatný MAC.
decompression_failure	30	Selhání dekompresní funkce kvůli špatnému vstupu.
handshake_failure	40	Nepodařilo se vyjednat bezpečné parametry pro spojení.
no_certificate	41	Klient informuje server, že nemá žádný certifikát. (pouze SSL)
bad_certificate	42	Upozornění na neověřitelný certifikát.
unsupported_certificate	43	Upozornění na nepodporovaný certifikát.
certificate_revoked	44	Upozornění na zrušený certifikát.
certificate_expired	45	Upozornění na neplatný certifikát.
certificate_unknown	46	Nepřijatelný certifikát kvůli nespecifikované chybě.
illegal_parameter	47	Nekonzistence v parametrech při vyjednávání spojení.

1.2.5 Protokol aplikační vrstvy

Protokol aplikační vrstvy (*SSL/TLS Application Data Protocol*) slouží k přebírání dat od aplikační vrstvy TCP/IP modelu a jejich předávání protokolu záznamové vrstvy SSL/TLS ve chvíli, kdy už je navázané spojení.

Kapitola 2

Útoky na SSL/TLS

V této kapitole jsou popsány útoky Heartbleed, DROWN, FREAK, Triple handshake a SWEET32. U každého útoku naleznete kromě obecných základních informací také popis principu, na jakém daný útok funguje, jeho možností, případných omezení a také způsobů, kterými je možné útoku zabránit.

2.1 Heartbleed

Heartbleed je jedním ze známějších útoků týkajících se protokolu TLS objevených v poslední době. Na rozdíl od některých jiných útoků se však nejedná o využití slabiny v principech protokolu, nýbrž chyby v jedné z jeho konkrétních implementací, a to v knihovně OpenSSL. Chyba umožňující tento útok byla objevena v roce 2014 a bylo jí přiděleno označení CVE-2014-0160.[2] První část názvu (Heart) získal podle procesu nazývaného heartbeat, se kterým útok úzce souvisí. Část bleed (v překladu krvácet) odkazuje na únik informací, který způsobuje.

2.1.1 Heartbeat

Proces nazývaný heartbeat byl do implementace TLS přidán v roce 2012. Umožňuje oběma stranám, které komunikují prostřednictvím protokolu TLS, kontrolovat funkčnost spojení. Nahradil tak starší způsob ověřování, který byl technicky náročnější. Kterákoliv strana spojení může kdykoliv zaslat té druhé tzv. *heartbeat request*. Ten sestává z krátké zprávy a informace o její délce. Druhá strana pošle zpět tzv. *heartbeat response*, která obsahuje identickou zprávu, a tím potvrdí, že je spojení stále funkční. V případě, že první strana neobdrží *heartbeat response*, je spojení ukončeno. Maximální možná délka heartbeat zprávy je 64 KiB. [3] [4]

2.1.2 Popis útoku

Konkrétní problém, který útok Heartbleed umožňuje, spočívá v chybné implementaci zprávy *heartbeat response*, kde nebylo žádným způsobem ošetřeno, zda informace o délce, kterou *heartbeat request* obsahoval, skutečně odpovídá délce dané zprávy. Do zprávy *heartbeat response* byla bez kontroly zkopírována požadovaná velikost dat z paměti. Chyba byla tedy způsobena pouze jedním neošetřeným voláním *memcpy(bp, pl, payload)*, kde *bp* je ukazatel na vytvářenou odpověď, *pl* ukazatel na místo, kde mají být data ke kopírování, a *payload* je požadovaná délka.[3][4] Útočníkovi díky tomu stačilo, aby v požadavku udal větší délku zprávy, než jakou ve skutečnosti poslal. Tím mohl od druhé strany snadno získat data, která se v paměti nacházela za samotnou zprávou získanou z *heartbeat request*. Útočník mohl při využití této slabiny získat až 64 KiB paměti cílového serveru.[2] To, jaká data získal, záleží čistě na momentálním obsahu paměti, ale potenciálně tak mohl zjistit například uživatelská jména, hesla, soukromé klíče, zprávy, nebo jakékoliv jiné citlivé informace. Útok mohl být navíc proveden vícekrát. [5]

Je několik důvodů, které napomohly tomu, aby útok Heartbleed byl ještě větším bezpečnostním rizikem. Jednou z jeho vlastností například je, že po sobě nezanechává žádné stopy v jakémkoliv logu a není tedy nijak detekovatelný. Dalším problémem bylo, že se poměrně rychle rozšířily zranitelné verze OpenSSL. Konkrétně se jednalo o verze 1.0.1 až 1.0.1f a 1.0.2-beta1. [2] Tyto verze implementovaly protokol TLS verze 1.2, který byl vyžadován z bezpečnostních důvodů kvůli dřívějším útokům, například útoku BEAST.

Útokem Heartbleed bylo potenciálně ohroženo až 66 % světových serverů. Takový byl totiž v roce 2014 podíl serverů Apache a Nginx, které využívají knihovnu OpenSSL.[6] Dále jsou dokonce ohrožena i jiná zařízení než webové servery, neboť Heartbleed je jedním z mála útoků, které mohou být provedeny ze strany serveru vůči klientovi. Zranitelné verze OpenSSL využívají například chytré telefony s operačním systémem Android verze 4.1.1 nebo směrovače firem Cisco a Juniper.[3] Konkrétní operační systémy, které byly nebo jsou vůči tomuto útoku zranitelné, jsou například Debian Wheezy, Ubuntu 12.04.4 LTS, CentOS 6.5, Fedora 18, OpenBSD 5.3, FreeBSD 10.0, NetBSD 5.0.2 nebo OpenSUSE 12.2.[5]

Zmíněná chyba umožňující útok byla opravena ve verzi OpenSSL 1.0.1g.[2] Oprava spočívá ve dvou přidaných kontrolách. První ověřuje, zda délka Heartbeat zprávy není nulová, pokud tento případ nastane, je zpráva hned odmítnuta. Druhá kontrola zajišťuje, že udaná délka skutečně odpovídá délce obdržené zprávy.[3]

Útoku lze zamezit aktualizací na bezpečné verze OpenSSL (1.0.1g a novější). V případě, že není možná aktualizace, je druhou možností překompilovat OpenSSL s přepínačem -DOPENSSL_NO_HEARTBEATS, což zabrání používání procesu heartbeat. [2] Útoku nijak nezabrání použití certifikátů, neboť heartbeat request lze poslat ještě před samotnou autentizací stran ko-

munikace. U systémů, které byly vůči útoku zranitelné, je po aktualizaci na potřebnou verzi vhodné změnit hesla a vytvořit nové soukromé klíče a certifikáty. [5]

2.2 DROWN

Název DROWN je zkratkou pro *Decrypting RSA with Obsolete and Weakened eNcryption*, což v překladu znamená dešifrování RSA pomocí zastaralého a oslabeného šifrování.[7] Tento útok byl objeven a zveřejněn v roce 2016 a bylo mu přiděleno kódové označení CVE-2016-0800.[8] DROWN umožňuje dešifrovat TLS spojení díky podpoře již nedostatečně bezpečného protokolu SSLv2 a kvůli úmyslně oslabeným exportním šifrám. Útok využívá princip tzv. *Bleichenbacher padding oracle attack*, při kterém útočník zasílá náhodné texty k dešifrování a díky výplním v šifrovaných blocích postupně získává informace o způsobu šifrování.[7][9] Tento typ útoku je již déle známý a výpočetně náročný. Pro jeho úspěšné provedení je potřeba milionů spojení. Chyby v SSLv2 ovšem způsobily, že se jeho náročnost snižuje na řády desetitisíců spojení.[7]

2.2.1 Bleichenbacher padding oracle attack

Tento typ útoku byl zveřejněn Danielem Bleichenbacherem v roce 1998. Využívá vlastností PKCS#1 v1.5, což je standard pro doplňování textu, který má být šifrován pomocí RSA.[7] Formát zprávy, která tomuto standardu odpovídá, je následující: 0x00 || 0x02 || PS || 0x00 || k, kde k je vlastní zpráva a PS je náhodně vygenerovaný vyplňovací řetězec o minimální délce jednoho bytu, který neobsahuje byte 0x00. [7][10]

Pokud má útočník zašifrovaný text c, který se chce pokusit dešifrovat, nejprve zjistí, zda server požaduje formát odpovídající PKCS#1 v1.5. To server obvykle prozradí buď odmítnutím neplatného textu, ukončením spojení nebo delším časem zpracování požadavku. V případě, že server zprávu přijme a dešifrovaný text m je tedy ve správném formátu, tak útočník ví, že pro něj platí $2B \leq m \leq 3B-1$, kde $B = 2^{8(|m|-2)}$. Pak se útočník pokusí vygenerovat nový šifrový text cn pomocí libovolného s tak, že $cn = (c \cdot s^e) \mathrm{mod} N$. Pokud není cn přijat, zkusí to útočník znovu s inkrementovaným s. Když najde vyhovující cn, tak ví, že pro určité r platí $2B \leq ms - rN \leq 3B$. To mu umožňuje omezit možný rozsah zprávy na $(2B + rN)/s \leq m \leq (3B + rN)/s$. Útočník pak zkouší vhodně odhadovat hodnoty r a s a postupně snižuje daný interval, až se dostane k jedné konkrétní hodnotě, tedy dešifrovanému textu. [7][10]

Tento útok je výpočetně poměrně náročný, vyžaduje miliony operací k dešifrování takového textu. Jedním z opatření proti tomuto útoku je zajištění toho, že útočník nezjistí, že je požadován text odpovídající PKCS#1 v1.5. V případě, že server obdrží text ve špatném formátu, tak pokračuje dál s náhodným falešným textem a ukončí spojení až ve chvíli, kdy je požadován *session key*, který útočník nezná. [7]

2.2.2 Exportní šifry

Exportní šifry byly zavedeny z bezpečnostních důvodů v 90. letech pro komunikaci opouštějící Spojené státy americké. Jedná se o šifry s úmyslně slabšími klíči, aby byla americká vláda v případě potřeby schopna komunikaci dešifrovat. Konkrétně například pro exportní RSA mohl mít klíč maximální délku 512 bitů.[11] Silnější šifry byly považovány za vojenskou zbraň. V době jejich zavedení se předpokládalo, že dostatečný výpočetní výkon pro jejich prolomení bude mít pouze americká NSA.[11] S postupem času a rozvojem technologií se však staly obecně nedostačujícími, jelikož dnes má i veřejnost přístup k prostředkům, které umožňují jejich prolomení. Omezení ohledně exportních šifer už byla převážně zrušena. Problém ale je, že nezmizela z implementací SSL/TSL.

2.2.3 Popis útoku

Útočník může využít server podporující SSLv2, který přijímá exportní šifry úmyslně oslabené délkou klíče, k postupnému dešifrování zachycené TLS komunikace. Tuto bezpečnostní slabinu způsobují tři vlastnosti SSLv2. První je skutečnost, že server okamžitě odpovídá zprávou *ServerVerify* ve chvíli, kdy od klienta obdrží zprávu *ClientMasterKey*, která obsahuje RSA šifrový text. A to bez toho, aby čekal na zprávu *ClientFinished*, která dokazuje, že klient zná původní text.[7] Druhým problémem je, že v případě, kdy je zvolena symetrická 40-bitová exportní RC2 nebo RC4 šifra, je pomocí RSA zašifrováno jen 5 bytů zprávy a zbylé byty jsou poslané nezašifrované.[7] Poslední vlastností, která tento útok umožňuje, je paradoxně opatření proti útoku na způsob *Bleichenbacher padding oracle attack*. Skutečnost, že server při nevalidním textu pokračuje v *TLS handshake* s náhodnými daty, ve výsledku stejně umožňuje útočníkovi rozpoznat platnost RSA šifrového textu právě díky náhodným datům, která se neshodují s původní zprávou.

Je více způsobů, jak útok provést. Jednou možností je, že útočník pasivně zachytí zhruba 1000 TLS handshaků využívajících RSA k výměně klíčů. Získané zašifrované zprávy se pak pokusí převést do formátu, který bude odpovídat PKCS#1 v1.5 pro server s SSLv2 a pomocí dříve popsaných vlastností takového serveru hledá validní RSA text, přičemž u každého textu musí hrubou silou prolomit slabou symetrickou šifru. V průměru se získá jeden platný text po 10 000 požadavcích. Jakmile útočník má takový validní RSA text, tak jeho drobnými úpravami prostřednictvím serveru s SSLv2 postupně pomocí dalších zhruba 10 000 operací získá celý původní text. [7]

Ve spojení s útokem DROWN, který je tedy umožněn chybami protokolu, byly však objeveny i některé implementační chyby v knihovně OpenSSL, které ještě více zvýšily šance k provedení útoku.[7][8] První chybou bylo, že servery s OpenSSL ve skutečnosti chybně podporovaly SSLv2 a exportní šifry, přestože byly výslovně nastaveny, aby taková spojení neumožňovaly. Této sa-

mostatné chybě byl přidělen kód CVE-2015-3197 a byla opravena ve verzích 1.0.2f a 1.0.1r. Další chyby, označované CVE-2016-0703 a CVE-2016-0704, umožňovaly provést ještě silnější verzi útoku, která byla funkční i proti neexportním šifrám. Dovolovaly, aby zpráva *ClientMasterKey* obsahovala nulové byty, ty pak byly nahrazovány byty zašifrovaného klíče. Útočník se tak mnohem snáze dostane přímo ke klíči. Tyto chyby byly neúmyslně opravené v roce 2015, tedy ještě před jejich objevením, díky změně, která opravovala jiný, nesouvisející problém. Speciální verze útoku DROWN využívající tyto chyby byla dostatečně rychlá na to, aby nemusela pouze dešifrovat stará spojení, ale umožňovala MITM (*man in the middle*) útok na právě probíhající spojení.[7]

V době objevení útoku bylo vůči němu zranitelných zhruba 33 % serverů a to i přesto, že část z nich SSLv2 nepodporovala.[7] Problém je, že služby mezi sebou často sdílejí RSA klíče. Stačí tedy, aby jeden článek v komunikaci podporoval zastaralý protokol, a pak jsou zranitelná všechna související spojení. Nejčastěji byly jinak zabezpečené servery zranitelné prostřednictvím využívání služby SMTP.[7]

V případě úspěšného provedení útoku může útočník získat jakoukoliv komunikaci mezi klientem a serverem, což zahrnuje například uživatelská jména, hesla, čísla kreditních karet, zprávy a jiné dokumenty. Pro zabezpečení serveru proti útoku DROWN je potřeba jednak zajistit, aby samotný server nepodporoval SSLv2, a dále aby nesdílel klíče s žádnými servery, které by ho podporovaly. Typicky mezi ně patří webové servery, SMTP, IMAP a POP servery a jiný software, který využívá SSL/TLS.[7][9]

2.3 FREAK

Útok FREAK byl zveřejněn 3. března 2015 a byl mu přidělen kód CVE-2015-0204. Jeho název vznikl ze zkratky pro *„Factoring RSA export keys“*, tedy faktorizace exportních RSA klíčů.[12] Pro umožnění tohoto útoku jsou zásadní dvě věci. První jsou samotné exportní šifry a druhou chyby v některých implementacích SSL/TLS, které umožňují tyto šifry přijmout jako dostatečné šifrování.

2.3.1 Popis útoku

O slabosti a prolomitelnosti exportních šifer se ví delší dobu, ale z několika důvodů nebyly považovány za velké nebezpečí. Jedním z nich je, že většina dnešních klientů, například webových prohlížečů, během zahajování komunikace s nějakým serverem vůbec nenabízí exportní šifry jako možnost. Dalším důvodem je předpoklad, že téměř žádné dnešní servery již takové šifry nepodporují. A posledním důvodem je, že i když je 512bitový klíč prolomitelný, stále to vyžaduje poměrně velký výpočetní výkon, než aby se to útočníkovi běžně vyplatilo.[11]

Problémem však byla chyba v některých implementacích SSL/TLS. Ta způsobovala, že klient mohl přijmout exportní RSA jako způsob šifrování, přestože ho sám nenabízel. To umožňuje provést MITM útok, při kterém útočník zajistí, aby klient a server pro komunikaci používali tuto slabší šifru. Průběh takového útoku je následující: [11][13]

1. Klient ve své *ClientHello* zprávě zašle serveru seznam šifer, které akceptuje pro komunikaci.
2. Útočník jako MITM tuto zprávu zachytí a pozmění ji tak, že nabízí pouze exportní RSA.
3. Server odpoví 512 bitovým exportním klíčem, který klient kvůli zmíněné chybě přijme.
4. Útočník provede faktorizaci klíče, aby získal dešifrovací klíč. S dešifrovacím klíčem pak může číst celou komunikaci a případně ji i pozměňovat.

Pro uskutečnění tohoto útoku je tedy kromě chyby v implementaci na straně klienta dále potřeba, aby server podporoval exportní šifry. Předpokládalo se, že takové servery už v podstatě neexistují. Ve skutečnosti se však krátce po zveřejnění útoku zjistilo, že 36,7 % světových serverů je podporuje. Patřily mezi ně i některé vládní weby, dokonce včetně www.nsa.gov.[11]

Posledním důvodem, proč tento útok představuje větší riziko, je, že útočník ve skutečnosti nemusí prolomit šifru během jednoho spojení a nepotřebuje tedy tak velký výpočetní výkon. Generování nového RSA klíče je poměrně náročné, a proto dnešní servery negenerují nový pro každé spojení. Ve skutečnosti často vygenerují jeden klíč při svém spuštění a ten pak používají pro více spojení. [11][13] Útočníkovi tak stačí klíč z jednoho spojení získat a použít ho až pro nějaké další, což mu dává více času na jeho faktorizaci.

Mezi knihovny, které umožňují využít této slabiny, patří například OpenSSL verzí 1.0.1j a starších, verze BoringSSL zveřejněné před 10. listopadem 2014, nebo verze SecureTransport společnosti Apple pro systém iOS verzí starších než 8.2 a pro systém OS X před opravou *Security Update 2015-002*. Mezi prohlížeče, které tyto knihovny využívají a jsou tedy zranitelné, patří například Google Chrome před verzí 41, Opera do verze 28, nebo Android Browser.[12] Pro zabezpečení proti útoku FREAK je potřeba aktualizovat na novější verze softwaru, které již obsahují opravu a jsou proti tomuto útoku zajištěné.

2.4 Triple handshake

Útok Triple handshake [14][15] byl objeven na začátku roku 2014. Jedná se o MITM útok, který útočníkovi ve výsledku umožňuje získat úplnou kontrolu nad určitým spojením. Tedy číst veškerou probíhající komunikaci, nebo i cíleně měnit její obsah. Tento útok není závislý na konkrétní implementaci,

ale využívá několika slabin samotného protokolu TLS. Jak jeho název napovídá, jedná se o slabiny související s procesem navázání spojení (v angličtině *handshake*), využívajícího k dohodě klíčů buď algoritmus RSA, nebo Diffie-Hellman. Výsledkem takového útoku je, že pokud se klient nevědomě připojí k nebezpečnému serveru, ten pak může předstírat, že je daným klientem u jiného serveru.

Pro tento útok jsou zásadní čtyři vlastnosti či slabiny protokolu TLS:

- V případě navazování spojení s využitím RSA klient zasílá serveru tak zvané *premaster secret*, což je náhodně vygenerovaná hodnota využitá pro algoritmus RSA. V případě, že se klient připojí k nebezpečnému serveru (útočníkovi), ten následně použije získané *premaster secret* k navázání spojení s dalším serverem. Ve výsledku tak útočník může synchronně udržovat obě spojení.

- V případě použití DHE server volí parametry pro algoritmus Diffie-Hellman. Pokud se stejně jako v předchozím bodě jedná o nebezpečný server, tak může zvolit neprvočíselné parametry. Díky tomu má útočník pod kontrolou *premaster secret*, které z nich klient vypočte, a může se pomocí něj již opět pomocí RSA připojit k třetímu serveru a dostat se do stejné situace, jako v předchozím bodě.

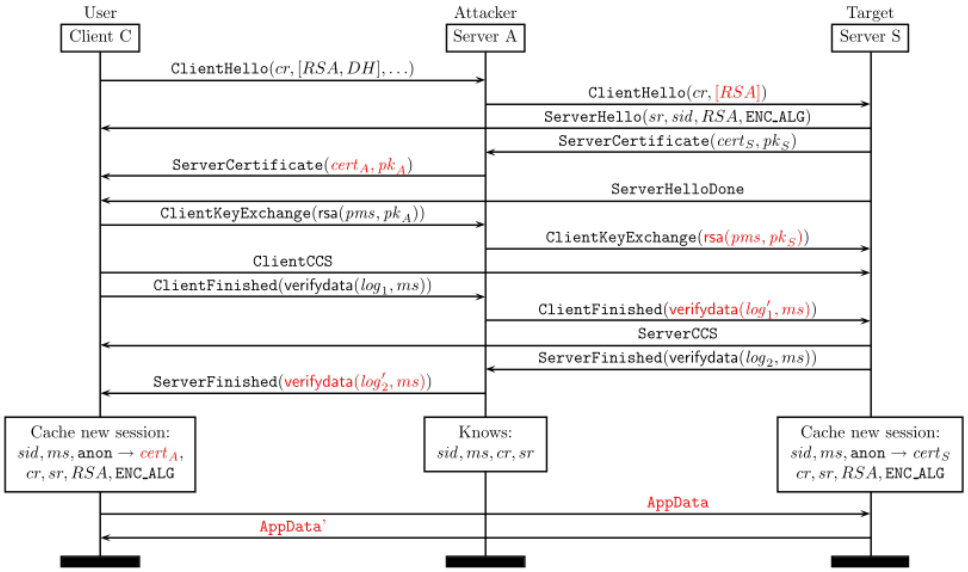
- Pro obnovení již dříve uskutečněného spojení se používá pouze zkrácená varianta procesu. Zkontroluje se pouze, zda strany sdílejí stejné *master secret*, nastavenou šifru a SID. Pro útočníka je tak poměrně snadné obnovit výše zmíněný stav spojení, pokud už se mu podařilo dříve tyto informace získat.

- Poslední podstatnou vlastností je skutečnost, že TLS z principu dovoluje, aby se při znovuvyjednání parametrů spojení změnily certifikáty klienta či serveru. Není však nijak určeno, jak v takové situaci postupovat.

2.4.1 Popis útoku

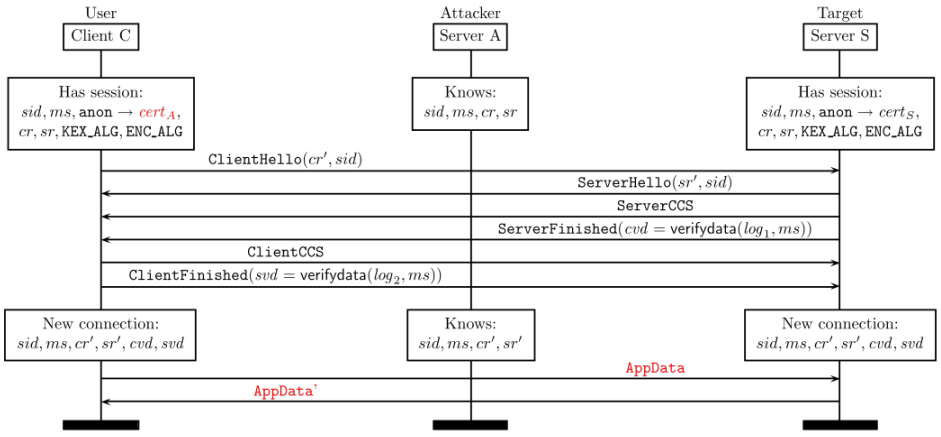
V případě, že chce útočník A provést tento útok na spojení mezi klientem C a serverem S, kdy je zvoleno použití RSA, má útok několik částí, jejichž průběh je následující.

V první části se klient C úmyslně připojí k serveru A, aniž by věděl, že A je ve skutečnosti útočník. A se připojí k serveru S pomocí stejné zprávy *ClientHello*, kterou obdržel od C, pouze s tou změnou, že v ní jako podporovanou šifru nabízí už jen RSA. Následně přepošle *ServerHello* od S klientovi, přičemž veřejný klíč S nahradí svým vlastním. A pak od C přijme jeho *premaster secret*, které dešifruje a znovu zašifruje veřejným klíčem S a přepošle mu ho. V tuto chvíli mají obě spojení stejné SID a *master secret*, ale odlišné certifikáty serverů a zprávy *Finished*. Tato část je znázorněna na obrázku 2.1.



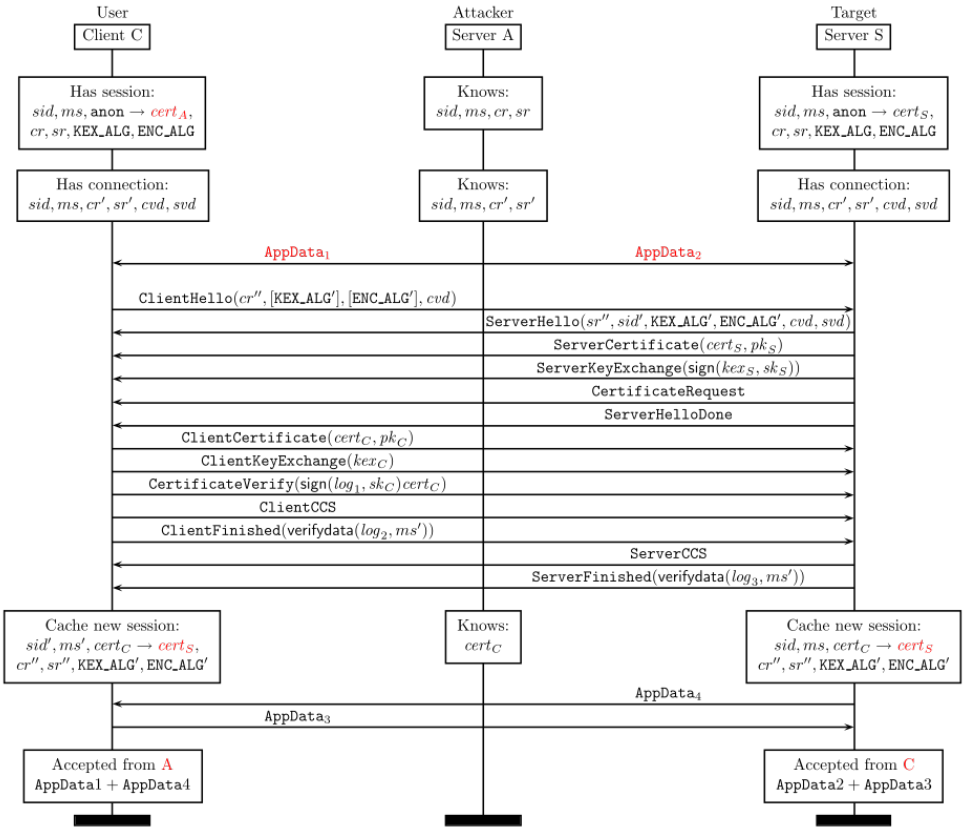
Obrázek 2.1: První část útoku [14]

Druhá část, znázorněná na obrázku 2.2, začíná ve chvíli, kdy klient C žádá A o obnovení spojení. A pomocí dříve získaných dat obnoví spojení s C a zároveň i s S. Po dokončení zkráceného procesu navazování spojení mají obě nové ale opět totožné klíče a nyní už i zprávy *Finished*. Útočník může číst nebo modifikovat veškerou komunikaci.



Obrázek 2.2: Druhá část při obnovení spojení [14]

Do této chvíle server S předpokládá, že je k němu připojen anonymní klient C. V momentě, kdy A požádá o přístup k nějakým zabezpečeným datům, si S vyžádá znovuvyjednání parametrů spojení kvůli autentizaci C. A tento požadavek opět přepošle, stejně jako následně přepošle zprávu s certifikátem od C. V této části C obdrží certifikát od serveru S, přestože očekával certifikát A, ale, jak je popsáno ve výše uvedených slabinách protokolu, záleží pouze na konkrétní implementaci, jak s touto podezřelou situací naloží a poměrně velké množství aplikací toto skutečně dovoluje bez žádného varování.[14] Tato poslední část je znázorněna na obrázku 2.3.



Obrázek 2.3: Třetí část při znovuvyjednání spojení [14]

Protože útočník nevlastní klientův certifikát, nemůže od této chvíle číst a modifikovat komunikaci, neboť nemá potřebné klíče. Spojení však může být stále ovlivněno. Útočník má například možnost ještě před třetí částí vložit do upravených zpráv pro klienta libovolný JavaScript, který se bude nadále vykonávat i po autentizaci klienta.

Tento útok lze provést, i pokud klient a cílový server podporují pouze použití DHE. V takové situaci serveru volí parametry pro jeho výpočet. Ve chvíli, kdy se C připojí k A, A se zároveň připojí k serveru S, od kterého získá zvolené parametry p a g a veřejný klíč $P_S = g^{K_S} \ mod \ p$. A však klientovi přepošle upravenou skupinu parametrů. Číslo p nahradí číslem $P_S(P_S - 1)$, což má za následek, že nezávisle na klíči klienta $P_C = g^{K_C} \ mod \ P_S(P_S - 1)$ bude výsledné *premaster secret* rovné číslu P_S:

$$\begin{aligned} premaster\_secret &= P_S^{K_C} \ mod \ P_S(P_S - 1) \\ &= P_S \ mod \ P_S(P_S - 1) \\ &= P_S \end{aligned}$$

Útočník ještě zašle serveru S g jakožto veřejný klíč a obě spojení mají opět (jako v předchozím případě) stejné *premaster secret*, *master secret* a klíče.

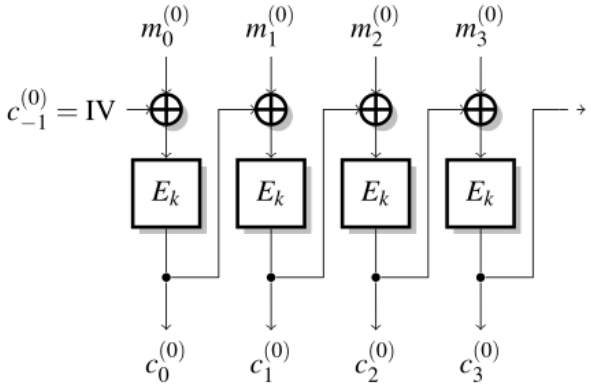
Jedním ze způsobů, jak tento útok částečně omezit, je upravit implementace SSL/TLS tak, aby klient pečlivě kontroloval platnost a příslušnost certifikátů, které přijímá. Ideálně také aby ukončil spojení v případě, že dojde ke změně certifikátů. Tato úprava však zabraňuje pouze poslední části útoku. Nejspolehlivějším řešením by bylo dosáhnout toho, aby nebylo možné navázat dvě různá spojení pomocí identického *master secret*. Jedna možnost, jak toho docílit, je například zahrnutí hodnoty hash vypočítané z vlastní zahajující zprávy do výpočtu náhodné hodnoty *premaster secret*. Tak by byl obsah *master secret* závislý na identitě klienta a serveru i na parametrech samotného spojení. Dalším způsobem, jak možnost útoku zkomplikovat, i když ne zcela vyloučit, je zvýšení bezpečnosti procesu zkráceného obnovení spojení například pomocí použití hashů původních zpráv *ClientHello* a *ServerHello* použitých při prvním navázání spojení.

2.5 SWEET32

Útok SWEET32 [16] byl zveřejněn 24. srpna 2016 a bylo mu přiděleno kódové označení CVE-2016-2183. Je to takzvaný narozeninový útok na blokové šifry v TLS, které používají při šifrování dnes již nedostačující 64-bitové bloky. Takovými jsou například DES nebo triple-DES. Termín narozeninový útok je odvozený od známého narozeninového problému. Ten se zabývá pravděpodobností, že v určité skupině lidí je více než 50% pravděpodobnost, že dva lidé mají narozeniny ve stejný den. Při převedení tohoto problému do oblasti šifrování zjistíme, že pro 50% pravděpodobnost nalezení takové kolize je potřeba $2^{n/2}$ operací, kde n je délka bloku šifry. V případě 64-bitového bloku tedy 2^{32} operací, odtud název útoku SWEET32.

2.5.1 Popis útoku

Jedná se o útok hrubou silou, jehož cílem je najít blok textu, který po zašifrování odpovídá získanému šifrovanému. Teoretická možnost těchto útoků je samozřejmě dobře známá, ale dlouhou dobu nebyly považovány za praktické a reálně proveditelné. Očekává se, že blokové šifry o blocích délky n budou odolné proti 2^n operacím. U části používaných šifrovacích módů, v tomto případě konkrétně u módu CBC (*Cipher Block Chaining*) je však tato složitost pouze $2^{n/2}$.



Obrázek 2.4: Znázornění šifrovacího módu CBC [16]

Šifrovací mód CBC, který je znázorněn na obrázku 2.4. je jedním z nejstarších stále používaných způsobů šifrování. Při jeho použití se zpráva M rozdělí na bloky m_i, kde $0 \leq i < l$ a tyto bloky jsou zašifrovány podle vzorce $c_i = E_k(m_i \oplus c_{i-1})$, přičemž c_{-1} pro první blok je zvolená hodnota nazývaná inicializační vektor (IV). Pro dostatečnou bezpečnost šifrování je vhodné, aby pro každou novou zprávu byl použit jiný IV, který nemůže případný útočník odhadnout. Mód CBC je prokazatelně bezpečný do množství $2^{n/2}$ zašifrovaných bloků.[17] S dalšími bloky zašifrovanými stejným klíčem však rychle narůstá šance nalezení kolizí. Kolize mezi šifrovými bloky znamená i stejný obsah před zašifrováním, tedy že pokud $c_i = c_j$, tak $m_i \oplus c_{i-1} = m_j \oplus c_{j-1}$. Díky tomu lze na základě znalosti šifrových bloků určit část informací o obsahu bloků: $m_i \oplus m_j = c_{i-1} \oplus c_{j-1}$.

Samotná znalost hodnoty xor dvou bloků ještě nemusí být pro praktický útok dostačující, za určitých podmínek se však tato situace podstatně mění. Pokud jsou nějaká tajná data posílána opakovaně a útočník zná zlomek původního obsahu, vzniká šance, že nastane kolize mezi bloky s tajnými a známými daty, což umožňuje útočníkovi snadno dešifrovat tajná data. Při provedení takového útoku by útočník nejprve zachytil a uložil všechna posílaná data a následně našel kolize mezi jednotlivými bloky. V případě, že útočník zná

pozice tajných dat a známého textu, snadno určí, které kolize jsou pro něj užitečné, a následně již zmíněným postupem dešifruje tajná data. V případě, že je útočník schopný například pomocí vhodného javascriptového kódu ovlivnit spojení mezi klientem a serverem tak, aby trvalo dostatečně dlouho, může tímto způsobem získat různé citlivé informace včetně přístupových hesel.

Šifry s 64-bitovými bloky jako například Triple-DES jsou stále používány a podporovány protokoly TLS nebo SSH. V implementacích protokolu TLS je zmíněná šifra dokonce povinnou součástí a podporuje ji tak zhruba 87 % serverů.[18] Tyto šifry jsou v protokolech stále udržovány kvůli zpětné kompatibilitě se staršími neaktualizovanými klienty, kteří nepodporují novější šifrovací algoritmy jako například 128-bitový AES. Nejbezpečnějším opatřením proti tomuto útoku je samozřejmě slabé šifry vůbec nepoužívat, pokud je to možné. V případě, že se jejich použití nelze vyhnout, je možné šance na útok minimalizovat dostatečně častým obnovováním šifrovacího klíče. To znamená zavádění nového klíče dříve než po $2^{n/2}$ blocích. Dalším opatřením je nastavení serverů tak, aby preferovaly 128-bitové šifry a Triple-DES používaly pouze s klienty, kteří nenabízejí žádnou jinou možnost. V souvislosti se zveřejněním tohoto útoku například knihovna OpenSSL ve verzích 1.0.1 a 1.0.2 přemístila šifru Triple-DES ze skupiny vysoce bezpečných mezi středně bezpečné a ve verzi 1.1.0 bude její použití potřeba úmyslně povolit.[16]

Kapitola 3

Demonstrace útoku

Obsahem praktické části této bakalářské práce je vytvoření aplikace, která bude názorně demonstrovat některé z dříve popsaných útoků. Konkrétně se bude zabývat dvěma variantami útoku Heartbleed. Tedy jak častější formou útoku, kdy je cílem server, tak i méně obvyklou variantou, která cílí na klienta. Důkladnější zaměření na tento útok bylo zvoleno především ze dvou důvodů. Prvním je jeho nenáročnost na použitý hardware, naproti tomu jiné útoky, jako například DROWN, jsou založené na prolamování šifrovacích a jiných kryptografických algoritmů, což má za následek výrazně větší výpočetní náročnost. S tím úzce souvisí druhý důvod. U útoků DROWN, FREAK či SWEET32 se jedná především o teoreticky zneužitelné slabiny v bezpečnosti, ale v případě útoku Heartbleed je díky nízké výpočetní náročnosti velice reálná šance, že se uživatel může stát cílem takového útoku. Vzhledem k tomu, že útočník může prostřednictvím Heartbleedu získat informace jako například hesla k účtům či jiná soukromá data, je o to důležitejší informovanost uživatelů o existenci a nebezpečnosti těchto rizik. Dále je potřebné informovat běžné uživatele nebo například i administrátory serverů o možnostech, jak takové situaci předejít. Aplikace, které jsou výsledkem této práce, tak lze použít i pro testování zranitelnosti jiných programů a serverů vůči tomuto útoku.

3.1 Zvolené technologie

Pro tuto implementaci bylo potřeba zvolit vhodné technologie v několika oblastech. První z nich je programovací jazyk, ve kterém bude aplikace napsána. V případě této práce byl vybrán programovací jazyk C. Tento jazyk byl zvolen především z toho důvodu, že je v něm zároveň napsána také knihovna OpenSSL, se kterou Heartbleed úzce souvisí, vzhledem k tomu, že je umožněn chybou v její implementaci. Díky stejnému jazyku je pak snadné využívat části kódu knihovny OpenSSL, které je možné po úpravě zneužít přímo k provedení útoku.

Dalším významným aspektem je vybraný operační systém. Původně byl vybrán systém Ubuntu verze 12.04, který byl vydán se zranitelnou verzí OpenSSL a podle zveřejněných informací popsaných dříve v této práci by tento systém měl být vůči tomuto útoku zranitelný. První pokusy o realizaci útoku však nebyly na tomto systému úspěšné, a proto byl ve výsledku zvolen systém Arch Linux v 32bitové verzi. Jeho výhodou je snadné downgradování na potřebné starší verze softwaru, které jsou vůči útoku Heartbleed zranitelné.

Co se týče zmíněných verzí softwaru, je především podstatná zvolená verze knihovny OpenSSL. Jak bylo zmíněno dříve, zranitelné jsou verze 1.0.1 až 1.0.1f. K demonstraci útoku byla využita poslední z nich, tedy OpenSSL verze 1.0.1f. U dalších používaných programů byly vybrány verze, jejichž datum zveřejnění je pokud možno blízké tomu u verze OpenSSL. Důvodem bylo vyloučení možnosti, že by byly dané programy závislé na funkcionalitách OpenSSL vyšších verzí. Konkrétně se tedy jedná o Nginx verze 1.4.4, curl verze 7.36.0 a wget verze 1.15. Ke kompilování kódů pak byl použit kompilátor gcc verze 6.3.1.

3.2 Existující řešení

První oficiální zprávy o útoku Heartbleed pocházejí z roku 2014. Existují tak již některé aplikace, které se jím zabývají. Naprostá většina takových implementací, které jsou k nalezení volně na internetu, se dá rozdělit do dvou skupin. První jsou testery, sloužící ke kontrole konfigurace webového serveru a zjištění, zda je vůči útoku zranitelný. Neprovádí tedy samotný Heartbleed. Druhá skupina jsou již implementace vlastního útoku. Většina z nich se však zabývá pouze obvyklejším útokem klienta na server.

3.3 Heartbleed cílený na klienta

Většina popisů a vysvětlení útoku Heartbleed se zabývá variantou, kdy je cílem útoku zranitelný server. V případě tohoto útoku je však stejně dobře uskutečnitelná i opačná verze, tedy že server může útočit na klienta, jakmile se klient připojí. Tato varianta je první, kterou se praktická čast práce zabývá.

3.3.1 Výběr řešení

Pro tuto implementaci je zásadní několik oblastí, ve kterých bylo potřeba zvolit vhodné řešení. První z nich je určení, na jaký typ klienta bude útok cílit. Možností by bylo vytvořit vlastního základního klienta komunikujícího pomocí TLS. Touto cestou by bylo možné klienta přizpůsobit požadovanému účelu, ale mohlo by být diskutabilní, do jaké míry by takové řešení ukazovalo skutečnou nebezpečnost útoku. Jelikož cílem této práce je demonstrování reálnosti a nebezpečnosti takového útoku, bylo zvoleno použití standardních

a neupravených klientů využívajících TLS. Pro tento útok je také zásadní, aby takový program využíval pro komunikaci knihovnu OpenSSL. Jelikož nejčastěji používané internetové prohlížeče využívají jiné implementace protokolu SSL/TLS, byly pro testování zvoleny programy curl a wget.

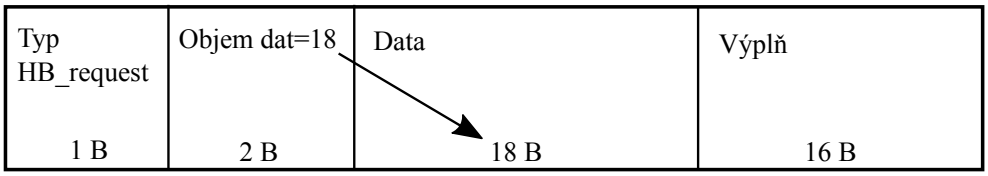
Další otázkou je vyřešení způsobu komunikace. Pro uskutečnění útoku je potřeba zaslat upravený heartbeat request. Jelikož se jedná o vnitřní proces v TLS, část existujících implementací řeší vytvoření potřebné zprávy jejím ručním sestavením z odpovídajících bytů a následným zasláním do socketu. Díky otevřenosti knihovny OpenSSL je ale možné použít i jiný způsob. Kvůli možnosti přístupu k jejím zdrojovým kódům je totiž možné pro útok zneužít přímo některé její vnitřní funkce. V takovém případě je možné změnit jen potřebné parametry zprávy, zatímco funkce zajistí zbylé náležitosti pro dodržení protokolu TLS.

Takto upravenou zprávu pak lze zaslat v různých fázích spojení. V případě této práce je předpokládána situace, ve které se uživatel nezodpovědně připojí k serveru s nedůvěryhodným certifikátem. Heartbeat request je v tomto případě zaslán až po dokončení handshaku. Nicméně útočník si může bez větších obtíží zařídit důvěryhodný certifikát a tím by snadno odpadla tato možná překážka v provedení útoku. Jinou možností by bylo zakomponovat útok přímo do procesu navazování spojení. V takovém případě může být útok úspěšný i bez platného certifikátu, protože únik dat se uskuteční ještě před jeho ověřováním.

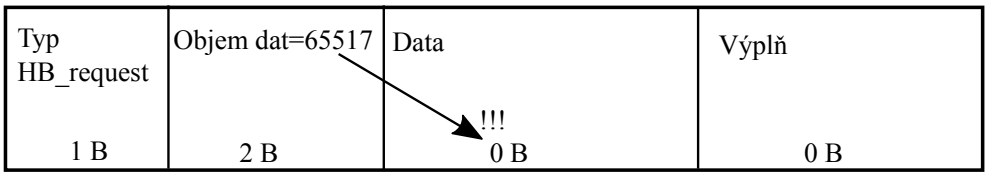
3.3.2 Popis implementace

Jako základ pro implementaci byl použit jednoduchý vzorový TLS server zveřejněný přímo na webu OpenSSL. [19] Z kódu tohoto serveru bylo potřeba odstranit pouze nastavení eliptických křivek, neboť tato funkcionalita byla přidána až v pozdějších verzích OpenSSL. Zároveň bylo přidáno omezení, které zajišťuje, že spojení nebude využívat protokoly SSLv2 a SSLv3, neboť proces heartbeat potřebný pro útok Heartbleed byl přidán až do protokolu TLS.

Další úpravy již souvisejí přímo s realizací útoku Heartbleed. Jak bylo již dříve zmíněno, byly využity přímo funkce ze zdrojového kódu OpenSSL. První z nich je funkce *tls1_heartbeat* ze zdrojového souboru OpenSSL t1_lib.c. Tato funkce slouží k zaslání standarní zprávy typu heartbeat request. Pro její zneužití k útoku bylo potřeba upravit ji tak, aby nebyla generována a odesílána náhodná data, která jsou obsahem obvyklého heartbeatu. Zároveň byla změněna udávaná velikost objemu dat, který je požadován jako odpověď. Standardní heartbeat, jehož struktura je znázorněna na obrázku 3.1, požaduje jako odpověď 18 bytů dat, ale maximální možná celková velikost odpovědi je 64 kibibytů, přičemž 3 byty zabírá hlavička a 16 bytů je automaticky generovaná výplň. Výsledný požadovaný objem tedy implicitně odpovídá 64 kibibytům bez těchto 19 bytů, ale je možné tuto velikost změnit parametrem programu. Upravený požadavek použitý k provedení útoku je znázorněn na obrázku 3.2.



Obrázek 3.1: Standardní heartbeat request



Obrázek 3.2: Heartbeat request upravený pro útok

Druhou funkcí OpenSSL, která je využita v této implementaci, je *ssl3_get_record*. Tato funkce má ve své původní podobě za úkol čtení a dešifrování nově příchozích paketů. V případě, že obdrží zprávu typu heartbeat, zavolá funkci pro jeho standardní zpracování a následně vyvolá čtení dalšího paketu. V případě této implementace je však potřeba pouze pro přijímání zpráv typu heartbeat response. Proto není potřeba její velká část zabývající se jinými typy zpráv a pro tuto implementaci nepodstatnými kontrolami. Naopak byla přidána kontrola ověřující, zda je příchozí zpráva typu heartbeat response či nikoliv. Dále muselo být přidáno zpracování vlastní odpovědi, tedy načtení údajů o obdržené heartbeat response, ošetření příjmu více paketů v případě fragmentace odpovědi a zapsání získaných dat do souboru.

3.3.3 Popis činnosti programu

Samotný program útočného serveru nejprve načte argumenty zadané při spuštění. Ty udávají port, na kterém server poslouchá, počet heartbeat requestů, které mají být odeslány případnému klientovi, a objem požadovaných dat. Dále provede činnosti potřebné pro funkci HTTPS serveru, tedy vytvoření socketu, inicializaci knihovny OpenSSL, vytvoření kontextu a jeho konfiguraci. Při konfuguraci je kontext nastaven tak, aby neakceptoval protokoly SSLv2 a SSLv3, neboť tyto protokoly neobsahují zprávy heartbeat. Následně již v cyklu přijímá připojující se klienty.

Po úspěšném připojení klienta je otestována konfigurace, zda jsou povoleny zprávy typu heartbeat. V případě kladného výsledku pokračuje program

načtením dat ze socketu. Tato data nejsou pro činnost programu potřebná, ale před samotným útokem je potřeba vyprázdnit socket, aby z něj mohla později být přečtena zpráva heartbeat response.

Následuje vlastní cyklus, ve kterém je postupně zaslán daný počet potřebně upravených zpráv typu heartbeat request. Data získaná z odpovědí heartbeat response jsou ukládána do souborů jejichž počet odpovídá počtu přijatých odpovědí. Jelikož může být požadována odpověď o velikosti až 64 kB, dá se předpokládat fragmentace této zprávy. Jednotlivé fragmenty jedné odpovědi jsou zapsány do společného souboru. Pro tuto část jsou použity dříve popsané upravené funkce z knihovny OpenSSL.

Po ukončení spojení s klientem server čeká na další požadavek o spojení.

3.3.4 Použití programu

Pro činnost tohoto serveru je neprve potřeba mít k dispozici certifikát a soukromý klíč. Program předpokládá jejich umístění ve stejném adresáři, kde se sám nachází. Pro jejich správné použití musí být soubor certifikátu nazvaný *cert.pem* a soubor klíče *key.pem*. Pokud nemá uživatel nějaký certifikát k dispozici, lze tyto soubory vygenerovat mimo jiné prostřednictvím řádkového programu openssl například pomocí následujícího příkazu:

```
openssl req -x509 -newkey rsa:4096 -keyout key.pem\
            -out cert.pem -days 365
```

Následujícím krokem je kompilace. Ta se může lišit v závislosti na použitém operačním systému. V rámci této práce byl, jak bylo dříve řečeno, použit systém Arch Linux. Na tomto a na některých dalších unixových systémech lze provést kompilaci programu například pomocí následujícího příkazu:

```
gcc HBserver.c server.c heartbleed.c -o HBserver -lssl -lssl3 \
            -lcrypto
```

Výjimkou jsou systémy založené na distribuci Debian, respektive Ubuntu. Na těchto systémech je lehce odlišná varianta knihovny OpenSSL. V první řadě může být potřeba nainstalovat balík *libssl-dev*, který obsahuje vývojářské nástroje a umožňuje používat potřebné knihovny. Dále je pak potřeba pozměnit způsob linkování knihoven. Výsledný příkaz pro kompilaci pak může vypadat takto:

```
gcc HBserver.c server.c heartbleed.c -o HBserver \
            -Wl,-Bstatic -lssl -Wl,-Bdynamic -lssl3 -lcrypto
```

Při obou způsobech se dají očekávat varování kompilátoru způsobená použitím vnitřních funkcí OpenSSL, u kterých není předpokládané použití uživatelem. Alternativou pro kompilaci je využít makefile přiložený v elektronické verzi práce.

Při spouštění program očekává dva povinné argumenty a případně třetí volitelný. Prvním je číslo portu, na kterém bude server poslouchat. Druhým počet heartbeat requestů, které mají být zaslány každému klientovi připojujícímu se k serveru. Třetí, volitelný, určuje objem požadovaných dat v bytech. Pokud ten není zadán nebo je zadán špatně, použije se nejvyšší možná hodnota pro délku heartbeat zprávy. Server lze tedy spustit například následujícím příkazem:

```
./HBserver 4433 1
```

K takto spuštěnému serveru se lze klientem na stejném počítači připojit pomocí adresy *https://localhost:4433*.

Získaná data z každé přijaté zprávy heartbeat response jsou uložena do souboru s názvem *bleed_n*, kde *n* je takové číslo, aby soubor s daným názvem zatím v daném adresáři neexistoval.

3.3.5 Testování

Pro testování serveru realizujícího útok Heartbleed byly použity řádkové programy curl a wget. Zranitelnost těchto programů se dala předpokládat, neboť oba využívají zranitelnou knihovnu OpenSSL. Kromě samotné úspěšnosti útoku byly zkoumány rozdíly v chování těchto programů, povaha útokem získaných dat a s tím související vlastnosti samotného útoku. Útok byl proti oběma programům podle předpokladu úspěšný. Na obrázku 3.3 je vidět, že kromě náhodného nečitelného obsahu paměti lze získat i smysluplné informace.

```
((...D(&.u..W.5o.T.*....$..          u...X.K....G...j._..j......
@...N...W..........aP..Hc..          .G~*...hx5M................
~.$...%..‘. PY....q......’.          .s.....(... .....s.Y8......
..K.KzA......_...........,.          ....{...........)...x.s.h.s
M....."c.W..,.i&.....(...;p          ..........F.{9.r....0...*.
i.=...{.........(.Z.N..b..C          H........071.0...U....Telia
>"1c.jN.p.a....=6O...g..!9~          Sonera1.0...U....TeliaSoner
4........^{G............].(          a Root CA v10...07101812005
..G’.......................          0Z..321018120050Z071.0...U.
 ...:p..:p...;...;.User-Age          ...TeliaSonera1.0...U....Te
nt: curl/7.36.0..Host: loca          liaSonera Root CA v10.."0..
lhost:4433..Accept: */*....          .*.H............0.........
```

Obrázek 3.3: Příklad získaných dat z programů curl(vlevo) a wget(vpravo)

Chování klientů

V chování těchto programů při různých scénářích útoku jsou vidět rozdíly. Server po získání dat prostřednictím heartbeat odpovědí neodesílá klientovi žádnou webovou stránku. Na tuto skutečnost reagují tyto dva programy odlišně. Zatímco program curl ukončí spojení s informací, že neobdržel odpověď, wget se v takovou chvíli pokouší stále znovu o nové spojení. Tím pádem se opakovaně připojuje k serveru a umožňuje tak opakovaně provádět útok, i když je server nastavený na zaslání pouze jednoho požadavku.

Další odlišnost v chování těchto klientů nastává při nastavení serveru, aby posílal více heartbeat požadavků při jednom spojení. V případě programu curl jsou všechny zpracovány a jsou na ně odeslány odpovídající odpovědi s požadovaným množstvím dat. Wget odešle očekávanou odpověď na první z požadavků, na další však odpoví špatným typem zprávy a nelze tak při jednom spojení získat tímto způsobem více dat.

Analýza získávaných dat

Při zkoumání dat získaných při testování útoku byl velmi patrný rozdíl v jejich obsahu podle toho, od jakého klienta byla získána. Data od programu curl obsahovala výrazně větší množství prázdných úseků. Počet nenulových bytů získávaných z jednotlivých odpovědí programu wget byl zhruba trojnásobný (viz příloha C.1). Tato situace se však mění při poslání více požadavků při jednom připojení. Při více požadavcích v řadě se počet nulových bytů od programu curl snížil na podobnou hodnotu, jako u programu wget.

Hlavní otázkou při zkoumání získávaných dat ale bylo, jak velký obsah paměti je možné útokem reálně získat. Od obou programů byly pro testování získány vzorky za následujících rozdílných situací, které by mohly ovlivňovat obsah a místo v paměti:

- Data o stejném objemu získaná po sobě z jedné a více instancí klienta.
- Data o různém objemu získaná po sobě z jedné a více instancí klienta.
- Data o stejném i různém objemu získaná z více instancí klienta, přičemž mezi jednotlivými instancemi byla na systému prováděna jiná činnost.
- Data o stejném i různém objemu získaná z více instancí klienta, přičemž mezi jednotlivými instancemi byly programy curl a wget použity k plnění jiných požadavků.

Předpokládalo se, že například požadování různých velikostí objemu dat způsobí alokování různých míst v paměti. Soubory se získanými daty byly porovnávány a vyhledávaly se mezi nimi případné shody.

Prvním zjištěním bylo, že mezi daty od programu curl a wget nebyly nalezeny žádné shodné úseky. Při porovnávání dat od jednoho programu už je

však tato situace jiná. V závislosti na velikosti daných souborů (podle velikosti heartbeat odpovědi) se v případě programu curl shodovaly úseky dat až o velikosti 2,4 KiB, přičemž celkový počet shodných úseků dat se pohyboval do 10. Ve výsledku tak celková shoda dosahovala až dvou třetin nenulové části souboru. Při zmíněném opakovaném požadavku a snížení počtu nulových bytů se objem shod téměř nezměnil, čili v tomto případě dosahovala celková shoda přebližně třetiny nenulových dat.

U programu wget dosahoval největší shodný úsek dat dokonce velikosti přibližně 15 KiB a celkový počet shod se pak pohyboval také kolem 10. Vzhledem k velikosti shod se tak výsledná data shodovala až z pěti šestin (viz příloha C.2). Zčásti tento rozdíl pravděpodobně souvisí i se zmíněnou odlišností v počtu nenulových bytů. Tyto shodné úseky se často nacházely i na shodných pozicích v souboru, což vypovídá o pravděpodobném kopírování stejné části paměti. Mezi odpověďmi o rozdílné požadované velikosti se v některých případech velikost největšího shodného úseku sice snížila až na 0,8 KiB, naopak ovšem vzrostl počet shod až ke 30 a celková míra shody pak mohla klesnou k jedné šestině. Jednalo se tak pravděpodobně o proměnlivejší část paměti, nicméně stále se znatelnou mírou shod. Oproti očekávání nebyly tyto shody ovlivněny různou činností systému či programů samotných.

Výsledek

Vzhledem k tomu, že jsou uniklá data získávána prostým kopírováním z paměti, mohou být dříve popsaná zjištění ovlivněna řadou faktorů. Od vlastností samotných programů na pozici klienta, přes vlastnosti daného operačního systému až po skutečnost, že tento systém byl využit především k realizaci tohoto útoku a neprobíhal na něm větší rozsah jiné činnosti. Nicméně zjištěné informace a míra shod mezi soubory naznačují, že cílové programy pracují při zprácování heartbeat požadavků s jednou omezenou částí paměti a není tak pravděpodobně jednoduše možné tímto útokem získat kompletní obsah celé paměti systému, kde program běží. Obsah získané části paměti ovšem záleží na náhodě a je možné, že se v ní budou v jiném případě nacházet citlivé informace.

Ještě nebezpečnější by byla situace, kdy by uživatel využíval programy využívající OpenSSL, které by pracovaly více s citlivými informacemi. Například internetový prohlížeč, či mailový klient, které pracují s různými přihlašovacími údaji a dalšími daty. Nelze vyloučit, že by od takových programů mohl útočník v krajním případě získat například přímo přihlašovací hesla nebo části soukromé komunikace.

3.4 Heartbleed cílený na server

Tato část se věnuje implementaci druhé a dříve již vícekrát zmíněné obvyklejší variantě útoku Heartbleed, při které útočník útočí z pozice klienta na webový

server.

3.4.1 Výběr řešení

Oblasti, ve kterých je možné volit různé přístupy při implementaci, jsou do značné míry analogické s předchozí variantou útoku. Větší význam zde má volba webového serveru, proti kterému bude útok prováděn. Vzhledem k tomu, že pro Heartbleed je potřebný a zásadní jen zlomek ze všech funkcionalit klasického serveru, mohl by útok být testován proti základnímu serveru obsahujícímu jen potřebné části pro demonstraci. V takovém případě ovšem existuje možnost vzniku dalších bezpečnostních chyb při implementaci samotného serveru. V takovém případě by mohlo být zpochybnitelné, zda práce ukazuje skutečné vlastnosti útoku. Pro demonstraci této implementace bylo z tohoto důvodu zvoleno testování proti existujícím serverovým aplikacím. První z nich je základnější server, který je k dispozici přímo v řádkovém programu *openssl*. Druhý pak webový server *Nginx*.

Vytvoření potřebných zpráv pro uskutečnění útoku je realizováno stejně jako v předchozím případě využitím upravených funkcí z knihovny OpenSSL. Volba chvíle pro zaslání upravené zprávy heartbeat request zde již nepůsobí takový rozdíl, jelikož nejčastěji se certifikátem prokazuje pouze strana serveru a při útoku vůči němu nehraje větší roli, zda je útok proveden při navazování spojení nebo až po jeho navázání.

3.4.2 Popis implementace

Vzhledem k tomu, že proces heartbeat je v obou směrech (od klienta k serveru a od serveru ke klientovi) identický, je shodná i velká část implementace obou variant. Bylo potřeba funkce pro realizaci útoku zakomponovat do programu klienta. Jako základ pro tuto implementaci byl použit jednoduchý TLS klient [20] jehož příklady jsou běžně k nalezení na internetu. Pro samotnou realizaci útoku byly použity stejné funkce, které jsou detailněji popsány u předchozí varianty.

3.4.3 Popis činnosti programu

Stejně jako popsaná implementace, i vlastní činnost programu je v mnoha ohledech podobná předchozí variantě útoku. Pomocí argumentů je načtena adresa a port, kam se má klient připojit, počet zpráv heartbeat request, které mají být danému serveru zaslány, a objem požadovaných dat. Následně je vytvořeno a navázáno nové TLS spojení. Dále je otestováno, zda je povolené použití procesu heartbeat. Na rozdíl od předchozí varianty se na straně klienta v tuto chvíli nenachází žádná příchozí data v socketu a je možné hned pokračovat provedením samotného útoku. Tato část je obdobná jako u první varianty. Po odeslání daného počtu požadavků a zpracování přijatých odpovědí je ukončeno spojení a následně i vlastní program.

3.4.4 Použití programu

V případě klienta není potřeba generovat pro jeho činnost certifikát. Prvním krokem je tedy kompilace kódu. Ta je taktéž analogická s první variantou útoku a není potřeba zde dané příkazy a problematiku distribucí opakovat. Pouze je potřeba změnit kompilovaný soubor na *HBclient.c* a název výsledného programu například na *HBclient*.

Při spouštění program očekává tři povinné argumenty a případně čtvrtý volitelný. Prvním je adresa cílového serveru, druhým číslo portu, ke kterému se má klient připojit. Třetím počet heartbeat requestů, které mají být zaslány cílovému serveru a posledním volitelným je objem požadovaných dat v bytech. Pokud volitelný argument není zadán, použije se implicitní hodnota, což je nejvyšší možná velikost heartbeat odpovědi. Tohoto klienta lze tedy spustit například následujícím příkazem:

```
./HBclient 127.0.1.1 443 1
```

V tomto případě se tedy klient připojí k serveru s adresou 127.0.1.1 na portu 443 a pošle mu jeden upravený heartbeat request.

Data získaná z odpovědi jsou opět uložena do souboru s názvem *bleed_n*, kde *n* je takové číslo, aby soubor s daným názvem zatím v daném adresáři neexistoval.

3.4.5 Testování

Pro testování této varianty útoku Heartbleed byly využity dva servery. Základní server obsažený přímo v řádkovém programu openssl a také server Nginx. Oba tyto programy využívají knihovnu OpenSSL a jsou tak při odpovídajících verzích zranitelné vůči tomuto typu útoku. Stejně jako u testování předchozí varianty útoku bylo cílem zjistit případné rozdílné chování různých serverů a také jaké jsou vlastnosti útokem získávaných dat. Na obrázku 3.4 jsou ukázky dat získávaných od obou serverů.

```
>nginx.org</a>.<br/>.Commerc        .........n..._...m...o...c..
ial support is available at.        .^...k...j...i...l...f...g..
<a href="http://nginx.com/">        .e...p...h...b...a...]...d..
nginx.com</a>.</p>..<p><em>T        .‘...Z...Y...[...W...V...H..
hank you for using nginx.</e        .X...U...P...T...R...Q...\..
m></p>.</body>.</html>......        .S...N...M...L...O...J...I..
.......p...127.0.0.1 - - [06        .K...G..................-..
/Dec/2017:23:03:08 +0100] "G        .........?..................
ET / HTTP/1.1" 200 612 "-" "        .....F.......@..........’..
curl/7.36.0".............           .%.......(...+...)...*...X..
```

Obrázek 3.4: Příklad uniklých dat od serverů Nginx(vlevo) a openssl(vpravo)

Chování serverů

Na výše uvedeném obrázku je zřejmé, že v datech od serveru Nginx bylo možné najít například informace o dřívějších požadavcích od jiných klientů. Naopak od serveru openssl nebyla získána data, která by dávala na první pohled smysl. Jak je ale vidět na obrázku, místy v nich byl patrný určitý vzor. Každopádně nelze předpokládat, že se v dané paměti nebudou za jiných situací nějaká citlivá data nacházet. Na rozdíl od cílových programů u předchozí varianty útoku zde však nebyl žádný rozdíl ve vlastním chování serverů. Oba zpracovávaly požadavky nezávisle na jejich počtu a objemu požadovaných dat.

Analýza získávaných dat

I v případě serverů se vyskytl výrazný rozdíl v podílu nenulových bytů v získaných datech. Data od serveru Nginx obsahovala často více než desetinásobek nenulových bytů oproti datům od serveru openssl(viz příloha C.1). Tato situace se opět mění při opakovaném požadavku při jednom spojení. V takovém případě se počet nenulových dat od serveru openssl přibližně ztrojnásobil. Naopak hlavním aspektem, ve kterém se tato forma útoku liší od předchozí testované, byla znatelně výraznější proměnlivost mezi postupně získávanými daty.

Pro zjištění, do jaké míry je tato forma útoku nebezpečná, byla od obou serverů opět pro testování získána data za následujících různých situací, které mohly ovlivnit stav nebo využívání paměti programů:

- Data o maximálním objemu získaná postupně po jednom požadavku.
- Data o různém objemu získaná postupně po jednom požadavku.
- Data o různém objemu získaná více požadavky najednou při jednom spojení se serverem.
- Výše zmíněné způsoby, přičemž mezitím byly na servery zasílany požadavky z jiných programů.

Jednoznačně neexistovala žádná vzájemná shoda mezi daty ze serveru Nginx a openssl. Výstupy od serveru openssl obsahovaly výrazný podíl nulových bytů a v tomto případě v nich nebyly nalezeny žádné čitelné smysluplné informace. Shodné úseky mezi jednotlivými daty získanými z tohoto serveru dosahovaly obvykle délky nanejvýš 0,3 KiB, celkový počet shod mezi dvěma výstupy se pohyboval od 0 do 2 (viz příloha C.2). Takto malé a málo četné shody mohly být způsobeny náhodou a nemuselo se jednat o stejné úseky v paměti. Při bližším porovnání dat je však patrné, že velké bloky nulových bytů se často nacházejí na podobných pozicích, což stejnou oblast v paměti naopak naznačuje. Zároveň nebyl získán vzorek dat, který by se povahou výrazněji odlišoval a při požadování více heartbeat odpovědí o menší délce při jednom spojení byl obsah odpovědí často identický.

Data získaná od serveru Nginx patřila z hlediska útoku k nejzajímavějším z obou variant útoku. Obsahovala menší počet nulových bytů a nacházely se v nich i čitelné úseky s informacemi například o předchozích požadavcích na server a o programech, které požadavek zaslaly. Informačně nejobsáhlejší výstupy byly získány při požadování velké až maximální velikosti heartbeat zprávy. Při požadování menšího objemu byly zřejmě alokovány jiné části paměti se značně větším počtem nulových bytů. Mezi jednotlivými získanými sadami dat se vyskytovaly shodné úseky o délce až 1,8 KiB a celkový počet shod se pohyboval mezi 10 a 20 (viz příloha C.2). V případě, že byl mezi jednotlivými útoky na server zaslán jiný požadavek, se délka shodných úseků snížila přibližně na 0,4 KiB. Pravděpodobně se tak jedná o podobné oblasti paměti, ale jejich obsah se z velké části mění.

Výsledek

Stejně jako u první varianty útoku mohou být získaná data ovlivněna systémem, na kterém je server spouštěn, či jinými faktory ovlivňujícími používání paměti. Nicméně ze zjištěných informací lze odhadovat, že ani touto formou útoku nelze postupně získat obsah celé operační paměti. Ovšem především u serveru Nginx bylo patrné, že je tímto útokem možné získat rozličné informace související s během serveru. To, k jakým informacím se může útočník dostat, pak záleží především na způsobu využití serveru.

3.5 Diskuze

V rámci síťových útoků je Heartbleed jednodušší, ovšem možná o to nebezpečnější příklad. Jiné útoky, z nichž některé jsou zmíněné i v této práci, jsou založeny například na prolamování složitých šifer, použití hrubé síly nebo mohou být závislé i na fyzickém přístupu k cílovému počítači. Uskutečnění takových útoků vyžaduje často rozsáhlé vědomosti, dostupný velký výpočetní výkon či dosti specifické situace. Od těchto se Heartbleed naprosto odlišuje. Pro jeho realizaci stačí navázat standardní spojení pomocí TLS protokolu, zaslat správně upravený požadavek a útočník bez dalších obtíží získá data oběti. Je pouze potřeba, aby cílový klient pro realizaci protokolu TLS využíval zranitelnou verzi knihovny OpenSSL. Na druhou stranu není nikdy jisté, jaká data útočník v případě úspěchu získá.

Díky tomu, že je tato možnost útoku již oficiálně zveřejněna, se může kdokoliv pokusit o jeho realizaci. Zároveň je však menší šance narazit na server či klienta, který proti němu ještě není zabezpečený. Jen těžko lze odhadnout, zda byl tento útok někým odhalen a realizován ještě před jeho oficiálním zveřejněním. Takový útočník by musel důkladně pročítat zdrojové kódy knihovny OpenSSL a musel by si všimnout chybějícího ošetření, které útok umožňuje. Nicméně během dvou let, kdy se o této chybě nevědělo, na ni někdo mohl narazit stejně jako lidé, kteří ji později zveřejnili.

Ale i přes známost a relativní jednoduchost tohoto útoku je pro jeho realizaci potřeba nemalé množství informací a práce. V této práci byly zmíněny dvě možnosti, které lze k implementaci použít. První je zasílaní upravených dat přímo do socketu. Tento případ vyžaduje bezchybnou znalost potřebných částí protokolu TLS, aby byl útočník úspěšný. Tedy přesně vědět, jaký formát a obsah musí mít zasílaná data. Druhou možností je využití přímo funkcí OpenSSL jako v případě této práce. Tento postup vyžaduje znalost souvislostí mezi jednotlivými funkcemi OpenSSL. K tomu je potřeba porozumění velké části zdrojového kódu knihovny, který není vždy jednoduše srozumitelný, tím pádem i dobrou znalost daného programovacího jazyka a také vlastního protokolu TLS. Nicméně všechny prvky potřebné k realizaci útoku vlastně poskytuje samotné OpenSSL.

Otázkou je, zda je možné takovým útokům a chybám v budoucnosti zabránit a případně jakým způsobem. V rámci chyb v implementaci protokolu se základní řešení hned nabízí. Možností by byla důsledná několikaúrovňová kontrola každé nové či upravené části kódu a pečlivé ošetřování všech situací, které mohou nastat. Právě chybějící ošetření umožnilo útok Heartbleed.

Bezpečnost je ovšem velmi komplexní a složitou oblastí informatiky. Vyloučit lidské chyby při její realizaci je sice velmi žádoucí, ale stejně jako v jiných oborech ne vždy na 100 % možné. Jiné útoky jsou místo vlastních chyb způsobeny překonáním starších bezpečnostních systémů a postupů. Vzhledem k neustále se měnícím a vyvíjejícím technologiím je nejspíš nepravděpodobné, že by jednou existoval protokol či systém, který by byl naprosto spolehlivě bezpečný. Maximum, co lze pro předcházení útoků dělat, je pokud možno včasné nahrazování zastaralých bezpečnostních algoritmů a postupů i důsledná kontrola všech součástí používaných protokolů a jejich implementací.

Závěr

Cílem této práce bylo seznámit čtenáře s protokolem SSL/TLS, zajišťujícím bezpečnost na internetu, a především i s některými jeho chybami, nedostatky nebo možnými bezpečnostními riziky. Konkrétně byly v práci kromě popisu samotného protokolu také popsány a analyzovány útoky Heartbleed, DROWN, FREAK, Triple handshake a SWEET32. Byly vysvětleny principy, na nichž jednotlivé útoky fungují, příčiny jejich proveditelnosti, jejich následky a možnosti omezení. Součástí práce byla také demonstrace dvou různých variant útoku Heartbleed.

V rámci praktické části práce byly vytvořeny dvě aplikace sloužící k demonstraci útoku Heartbleed. První k realizaci útoku cíleného na klienta a druhá k útoku cílenému na server. Práce obsahuje popis funkčnosti a použití těchto aplikací a lze je tak použít nejen pro demonstraci útoku, jak byla předvedena, ale také k otestování zabezpečení jiných programů proti danému útoku. Obě aplikace byly testovány proti reálně používaným programům implementujícím klienty, respektive servery, využívajícím protokol TLS pomocí knihovny OpenSSL. Při tomto testování bylo názorně předvedeno a zhodnoceno, jak nebezpečný může tento útok být.

Vzhledem k neustále nově objevovaným bezpečnostním rizikům by tato práce mohla být do budoucna rozšiřována o analýzy dalších útoků a bezpečnostních chyb. Případně o další nástroje sloužící k jejich demonstraci či k testování zabezpečení jiných programů proti nim.

Literatura

[1] Opplinger, R.: *SSL and TLS: Theory and Practice*. ARTECH HOUSE, 2009, ISBN 978-1-59693-447-4.

[2] OpenSSL Security Advisory [07 Apr 2014]. Technická zpráva, duben 2014, [cit. 2017-02-21]. Dostupné z: https://www.openssl.org/news/secadv/20140407.txt

[3] Chandra, B.: A technical view of the OpenSSL 'Heartbleed' vulnerability. 2014, [cit. 2017-02-21]. Dostupné z: https://www.ibm.com/developerworks/community/wikis/home?lang=en#!/wiki/W746177d414b9_4c5f_9095_5b8657ff8e9d/page/A%20technical%20view%20of%20theOpenSSL%20%E2%80%98Heartbleed%E2%80%99%20vulnerability

[4] Limer, E.: How Heartbleed Works: The Code Behind the Internet's Security Nightmare. 2014, [cit. 2017-02-21]. Dostupné z: http://gizmodo.com/how-heartbleed-works-the-code-behind-the-internets-se-1561341209

[5] http://heartbleed.com/, [cit. 2017-02-21].

[6] April 2014 Web Server Survey. 2014, [cit. 2017-12-23]. Dostupné z: https://news.netcraft.com/archives/2014/04/02/april-2014-web-server-survey.html

[7] Aviram, N.; Schinzel, S.; Somorovsky, J.; aj.: DROWN: Breaking TLS using SSLv2. Technická zpráva, 2016, [cit. 2017-03-02]. Dostupné z: https://drownattack.com/drown-attack-paper.pdf

[8] OpenSSL Security Advisory [1st March 2016]. Technická zpráva, březen 2016, [cit. 2017-03-02]. Dostupné z: https://www.openssl.org/news/secadv/20160301.txt

[9] https://drownattack.com/, [cit. 2017-03-02].

[10] Bleichenbacher, D.: Chosen Ciphertext Attacks Against Protocols Based on the RSA Encryption Standard PKCS #1. [cit. 2017-03-04]. Dostupné z: http://archiv.infsec.ethz.ch/education/fs08/secsem/Bleichenbacher98.pdf

[11] Green, M.: Attack of the week: FREAK (or 'factoring the NSA for fun and profit'). [cit. 2017-03-15]. Dostupné z: https://blog.cryptographyengineering.com/2015/03/03/attack-of-week-freak-or-factoring-nsa/

[12] FREAK: Factoring RSA Export Keys. [cit. 2017-03-15]. Dostupné z: https://mitls.org/pages/attacks/SMACK#freak

[13] Heaton, R.: The SSL FREAK vulnerability explained. [cit. 2017-03-15]. Dostupné z: http://robertheaton.com/2015/04/06/the-ssl-freak-vulnerability/

[14] Bhargavan, K.; Delignat-Lavaud, A.; Fournet, C.; aj.: Triple Handshakes Considered Harmful: Breaking and Fixing Authentication over TLS. [cit. 2017-03-21]. Dostupné z: https://mitls.org/pages/attacks/3SHAKE

[15] Bhargavan, K.; Delignat-Lavaud, A.; Fournet, C.; aj.: Triple Handshakes and Cookie Cutters: Breaking and Fixing Authentication over TLS. Technická zpráva, 2014, [cit. 2017-03-21]. Dostupné z: https://mitls.org/downloads/tlsauth.pdf

[16] Bhargavan, K.; Leurent, G.: On the Practical (In-)Security of 64-bit Block Ciphers. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ACM, 2016, ISBN 978-1-4503-4139-4, [cit. 2017-04-14]. Dostupné z: https://sweet32.info/SWEET32_CCS16.pdf

[17] M. Bellare, J. K.; Rogaway, P.: The security of cipher block chaining. In *Advances in Cryptology — CRYPTO '94 - 1994*, Springer, Berlin, Heidelberg, 1994, ISBN 978-3-540-48658-9, s. 341–358, [cit. 2017-12-23].

[18] FEBRUARY 2016 SCAN RESULTS (INCOMPLETE). 2016, [cit. 2017-12-23]. Dostupné z: https://securitypitfalls.wordpress.com/2016/04/17/february-2016-scan-results-incomplete/

[19] https://wiki.openssl.org/index.php/Simple_TLS_Server, [cit. 2017-11-22].

[20] https://www.cs.utah.edu/~swalton/listings/articles/ssl_client.c, [cit. 2017-11-28].

Příloha **A**

Seznam použitých zkratek

AES Advanced Encryption Etandard

CBC Cipher Block Chaining

CVE Common Vulnerabilities and Exposures

DES Data Encryption Standard

DHE Diffie-Hellman Ephemeral

DROWN Decrypting RSA with Obsolete and Weakened eNcryption

FREAK Factoring RSA Export Keys

HTTPS Hypertext Transfer Protocol Secure

IETF TLS WG Internet Engineering Task Force TLS Working Group

IMAP Internet Message Access Protocol

MAC Message Authentication Code

MITM Man In The Middle

NSA National Security Agency

PCT Private Comunication Technology

PKCS Public Key Cryptography Standards

POP Post Office Protocol

SHA Secure Hash Algorithm

SID Security Identifier

SMTP Simple Mail Transfer Protocol

SSH Secure Shell

SSL Secure Sockets Layer

STLP Secure Transport Layer Protocol

TCP/IP Transmission Control Protocol/Internet Protocol

TLS Transport Layer Security

Příloha **B**

Popis algoritmu Diffie-Hellman a RSA

Diffie-Hellman

Algoritmus Diffie-Hellman [1] je jedním z prvních způsobů pro dohodnutí společného tajného klíče. V roce 1976 ho poprvé publikovali Whitfield Diffie a Martin Hellman.

Při použití tohoto algoritmu mezi subjekty A a B nejprve A zvolí prvočíslo m a celé číslo a, přičemž $0 < a < m$. Následně si A zvolí číslo $k_1 < m$, které je jeho soukromým klíčem. Vypočte $y_1 = a^{k_1} \bmod m$ a čísla a, m a y_1 odešle B. B si zvolí svůj klíč $k_2 < m$ a stejně jako A nyní vypočte $y_2 = a^{k_2} \bmod m$ a pošle zpět y_2. V posledním kroku oba vypočítají společný tajný klíč $K = y_2^{k_1} \bmod m = y_1^{k_2} \bmod m$.

Pro bezpečnost tohoto algoritmu je zásadní zvolení vhodných parametrů. To primárně znamená, že m musí být dostatečně velké prvočíslo a čísla y_1 a y_2 musí být určena dostatečně náhodným generátorem, tak aby je případný útočník nemohl uhodnout.

V praxi se tento algoritmus používá v několika variantách. Pro první se používá zkratka DH a v angličtině se označuje jako *fixed Diffie-Hellman*. V této variantě jsou některé parametry algoritmu pevně dané a jsou součástí certifikátů komunikujících stran. Další možností je *ephemeral Diffie-Hellman* označovaný jako DHE. Zde jsou všechny parametry vybrané náhodně pro větší jistotu unikátnosti výsledného klíče a jsou nějakým způsobem autentizované jejich odesílatelem. Poslední variantou je *anonymous Diffie-Hellman* neboli DH_anon. Tato možnost je podobná jako DHE, ale parametry nejsou žádným způsobem autentizované, což ulehčuje případný útok.

RSA

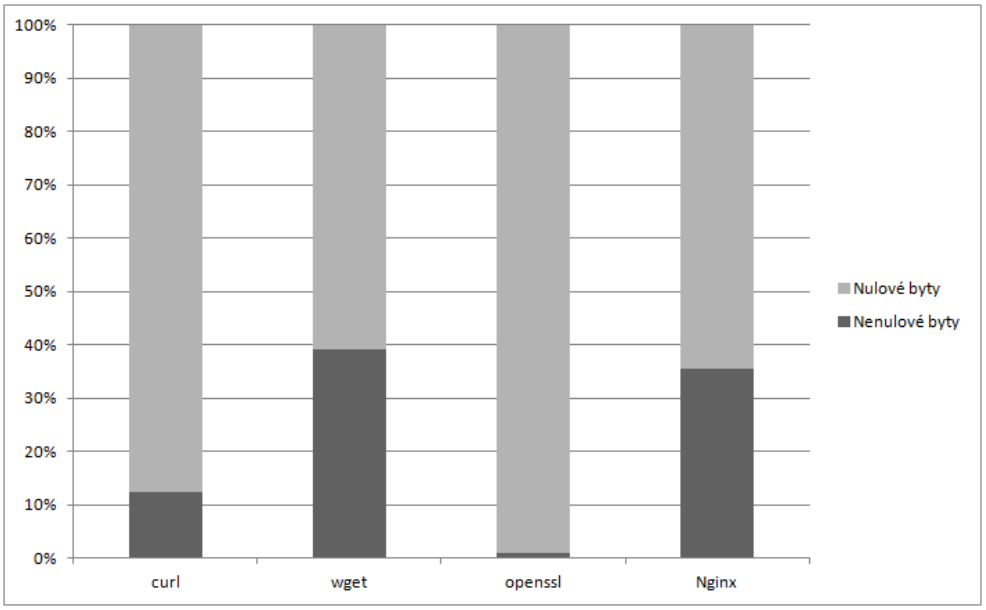
Algoritmus RSA[1] vyvinuli v roce 1977 Ronald Rivest, Adi Shamir a Leonard Adleman. Jedná se o asymetrickou šifru, kterou lze použít pro šifrování i podepisování dokumentů. Její bezpečnost spočívá ve složitosti faktorizace velkých čísel, tedy jejich rozkladu na prvočinitele.

Pokud chtějí subjekty A a B komunikovat prostřednictvím RSA, musí si nejprve oba vytvořit veřejný a soukromý klíč. Každý si zvolí náhodná velká prvočísla p a q. Následně vypočítá n, které je součinem p a q, a hodnotu Eulerovy funkce $\varphi(n)$. Dále si zvolí kladné celé číslo e, které je menší než n a nesoudělné s $\varphi(n)$. Poslední je potřeba určit d, které je číslem inverzním k e v modulu $\varphi(n)$. Dvojice (n, e) je pak veřejným klíčem a (n, d) klíčem soukromým. Postup šifrování pak je $c = me \bmod n$ a dešifrování $m = cd \bmod n$, kde m je číselný ekvivalent zprávy kterou chceme šifrovat, případně její blok, a c je odpovídající šifrový text.

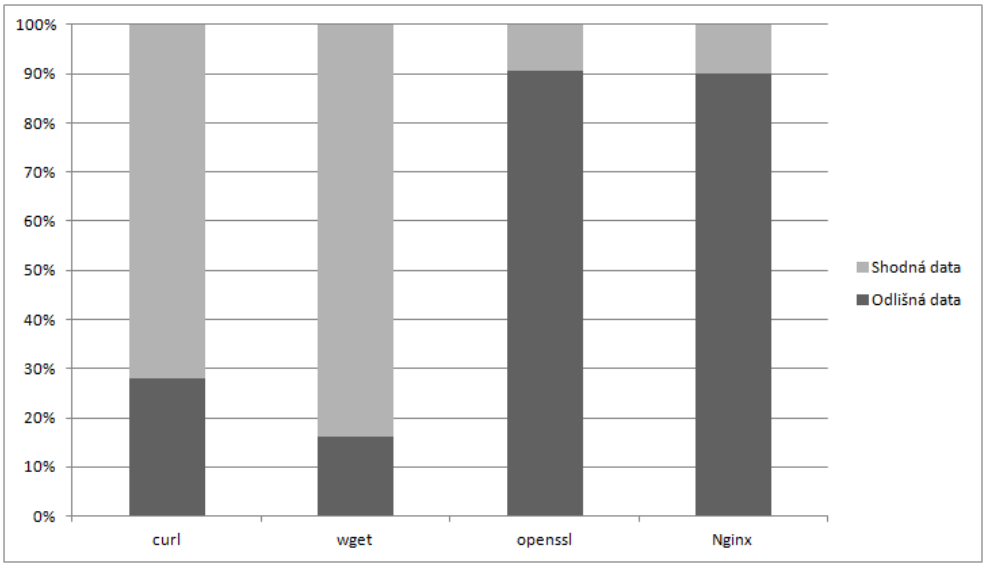
RSA lze využít i pro podepisování dokumentů. Pokud chce subjekt zprávu takto podepsat, připojí k ní zašifrovaný hash, k jehož zašifrování použil svůj soukromý klíč. Příjemce pak ověří podpis tím, že ho dešifruje veřejným klíčem odesílatele. Pokud získá odpovídající hash, má tak potvrzeno, že zpráva nebyla nijak upravena.

Příloha C

Znázornění výsledků testování útoku Heartbleed



Obrázek C.1: Podíl nulových a nenulových bytů v datech získaných útokem Heartbleed (při požadované maximální délce odpovědi po jednotlivých požadavcích)



Obrázek C.2: Podíl shod mezi nenulovými daty získanými útokem Heartbleed (při požadované maximální délce odpovědi po jednotlivých požadavcích)

Příloha D

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `data` ..... data získaná při testování útoku Heartbleed
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF