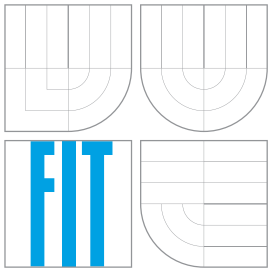
Vysoké učení technické v Brně
Brno University of Technology



Fakulta informačních technologií

Faculty of Information Technology

Projekt 1 - Tunel pre bezpečné zasielanie správ

Kryptografia

Autor práce Ladislav Šulák

Login xsulak04

Brno 04/2017

Úvod

Cieľom bolo vytvoriť aplikáciu, ktorá bude implementovať zabezpečený šifrovaný komunikačný kanál medzi dvoma uzlami. Program je možné spustiť v móde server alebo klient. Komunikácia je postavená na princípe *request-response*.

Implementačný jazyk bol zvolený C++, knižnica pre generovanie náhodných čísel (generuje z /dev/urandom) OpenSSL, ktorá bola tiež použitá pre šifrovanie. Okrem toho ešte knižnica GMP pre prácu s velkými číslami a tiež pre generovanie náhodných čísel, predovšetkým v počiatočnej fáze.

Detaily jednotlivých algoritmov, hlavne Diffie-Hellman (DH) a Feige-Fiat-Shamor, boli čerpané z doporučenej knihy Handbook of Applied Cryptography[1].

V nasledujúcich sekciách bude popísaný komunikačný protokol s detailnejším popisom zabezpečenia, analýza tohto zabezpečenia z celkového pohľadu, zopár implementačných detailov a na záver krátke zhrnutie.

Komunikačný protokol

Komunikačný kanál pre prenos dát medzi klientom a serverom bol postavený na unixových socketoch. Sůbor je uložený na */tmp/secured_kry1-xsulak04.socket*. Je teda pomerne jednoduché modifikovať aplikáciu tak, aby bola schopná prenášať dáta po sieti a server bol možný prijímať viacerých klientov, čo by nebolo v prípade pomenovaných rúr možné.

Na počiatku komunikácie si zašlú obe strany pozdrav, čo je nezašifrovaný text s využitím princípu *request-response*. Komunikácia teda začína na tomto mieste.

Následne sa ustanovuje tajný zdielaný kľúč pomocou metódy Diffie-Hellman, v ďalších krokoch využívaný v algoritme AES. Najprv sa vypočíta súkromný kľúč a následne sa vypočíta a predá verejný kľúč. Verejné modulo, nazývané aj ako *public prime modulo*, bolo získané z RFC3526[2], varianta *2048-bit MODP group*, pričom *prime base* bolo zvolené ako 2. Na základe týchto dát je možné vypočítať zdieľané tajomstvo. Pri predchádzajúcich činnostiach sa využíva generovanie náhodných čísel a funkcie operujúce nad veľkými číslami. O všetko sa starajú knižnice OpenSSL (tá v tejto časti iba pre generovanie seedu do funkcie *mpz_urandomm*) a GMP. Malo by ešte byť poznačené, že sa nekontroluje, či sa nevygeneroval slabý kľúč, k čomu rozhodne je možné dojsť. Tento algoritmus je náchylný na útok MitM, co vyplýva z povahy protokolu.

V ďalšom kroku dochádza k úprave dĺžky zdielaného kľúča na 256 bitov, pomocou hashovacej funkcii SHA256. Výsledok sa používa pre šifrovanie symetrickým algoritmom AES v režime CBC, ktorý nezaisťuje autentizáciu. Pri každom šifrovaní sa používa iniciačný vektor, generovaný znova pomocou knižnice OpenSSL. Nie je tajný, ale tým, že sa stále mení a má 128 bitov, ho nie je možné uhádnuť útokom silou.

V poslednej časti iniciovania zabezpečeného kanálu prebieha autentizácia pomocou Feige-Fiat-Shamirovho protokolu. Počíta sa v ňom síce s treťou stranou, ktorá distribuuje kľúče, no v rámci tejto práce boli hodnoty súkromného kľúča, resp. vektora $s0 - s4$ pridelené a sú umiestnené v zdrojovom kóde staticky. Overovanie prebieha v kolách, boli vybraté 4.

Po úspešnej autentizácii sa posiela klientovi správa o úspechu. Klient následne odosiela správu o svojom blízkom ukončení. Server mu odpovie hashom tejto správy (SHA256).

[1] http://cacr.uwaterloo.ca/hac/

[2] https://www.ietf.org/rfc/rfc3526.txt

Klient hash overí a obaja ukončia svoju činnosť.

Analýza protokolu

Aplikácia, resp. protokol, ktorý bol implementovaný, je odolný voči odpočúvaniu a ak by bola ešte vykonávaná kontrola na slabé kľúče, tak aj proti útoku hrubou silou. Čo sa týka útoku MitM, stačí, aby útočník v správny moment preposielal dáta a ani jedna z komunikujúcich strán nemá možnosť zistiť, že dáta preposiela niekto iný, v našom protokole ani v prípade FFS.

Implementácia

Projekt je implementovaný v jazyku C++. V zložke *services/* sa nachádzajú služby, implementujúce triedy pre prácu so socketami (*SocketService.h*), logovaním (*LoggerService.h*) a potom služba implementujúca bezpečnostné prvky spoločné pre klienta i server (*CryptoService.h*), teda AES, SHA256, posielanie zašifrovaných správ, vypočítanie kľúčov u Diffie-Hellmana, generovanie náhodných čísel a zopár ďalších pomocných metód.

V rámci tejto práce boli implementované aj testy v jazyku Python a Bash, nachádzajúce sa v zložke *tests/*, spústia sa pomocou *run_test.sh*. Nazerajú do výstupných logov a hľadajú, či sa vypočítal správne hash, zdielaný kľúč, korektnosť autentizácie a ďalšie. Logovanie do súborov je možné spustiť parametrom -t. Aplikácia naviac implementuje správanie serveru (parameter -s) a klienta (parameter -c), k čomu slúžia triedy Client a Server v príslušných .cpp a .h súboroch. Program je možné preložiť pomocou pribaleného s úboru *Makefile* a štandardne vypisuje svoj výstup na *stdout*.

Záver

V tejto práci sa podarilo implementovať výmenu kľúčov pomocou Diffie-Hellmana, šifrovať komunikáciu pomocou šifry AES256-CBC. Takisto sa podarilo využiť hashovací algoritmus SHA256 pre overovanie integrity správ i redukciu kľúča u DH.

Problém nastal u FFS protokolu, kedy sa nie vždy podarí, aby všetky kolá prebehli úspešne a teda v skutočnosti sa nepodarí autentizovať vždy, aj ked je aplikácia napísaná tak, že s server ani klient neskončia ale počítajú ďalšie kolá. Tento problém sa žiaľ nepodarilo vyriešiť úplne. Riešenie bolo vytvárané v prostredí OS Ubuntu 16.04 LTS 64-bit. Po prechod na školský server Merlin došlo k viacerým problémom, ako napríklad iné verzie modulov, čo viedlo k ťažšie odhalitelným chybám, prípadne neexistencia niektorých funkcií, hlavne z knižnice GMP, ktoré sa nachádzali v oficiálnej dokumentácií i v testovacom prostredí.