

POVEĆANJE STEPENA BEZBEDNOSTI VPN

Mladen Veinović

Univerzitet Singidunum

e-mail: mveinovic@singidunum.ac.rs

Dejan Savić

Vojna akademija

APSTRAKT: Virtuelne privatne mreže su prisutne u svakodnevnom životu. Realizuju se različitim tehnologijama od kojih svaka ima i prednosti i mane. Aspekt bezbednosti je jedan od ključnih detalja kojem se posvećuje puno pažnje pri njihovoj realizaciji. U ovom radu je predloženo poboljšanje bezbednosti virtuelnih privatnih mreža i izvršena je realizacija predloženog rešenja. Suština poboljšanja je u izmeštanju parametara za uspostavu VPN veze i pripadajućih kriptografskih ključeva na smart karticu, kao medijum koji je sigurniji od fajl sistema. Sa druge strane, čitanje parametara sa smart kartice je kompleksan zadatak za kriptografske aplikacije, tj. mnogo jednostavnije je čitanje fajlova. Iz ovih razloga, realizacija predloženog rešenja obuhvata kreiranje virtuelnog fajl sistema koji čita pomenute parametre sa smart kartice i omogućava njihovo korišćenje od strane aplikacije koja realizuje VPN vezu. Po prestanku korišćenja aplikacije virtuelni fajl nestaje iz memorije i ne ostavlja trag na memorijskim medijumima računarskog sistema. Izvršena je analiza realizovanih test aplikacija na *Linux/UNIX* mašinama, a korišćene su Java kartice.

Ključne reči: Virtuelne privatne mreže, virtuelni fajl sistem, *FUSE*, *PC/SC Lite*, Java Card

Virtual private networks are present in everyday life. Implemented by different technologies, each with advantages and disadvantages. Aspect of security is one of the key details which devotes much attention in their implementation. This paper proposes an enhanced security virtual private networks and made the implementation of the proposed solutions. The essence of improvement in the displacement parameters for establishing a VPN connection and associated cryptographic keys on smart cards, as a medium that is more secure than the file system. On the other hand, reading parameters from the smart card is a complex task for cryptographic applications, ie. much easier to read a file. For these reasons, the realization of the proposed solutions involve creating a virtual file system that reads the aforementioned parameters of the smart card and allows their use by the application that implements the VPN connection. Upon termination of use of application virtual file disappears from the memory and leaves no trace on the storage media of a computer system. The analysis of the realized test applications on Linux / UNIX machines, and used the Java card.

Key words: Virtual private networks, virtual filesystem, *FUSE*, *PC / SC Lite*, Java Card

1. UVOD

Virtuelne privatne mreže (VPN) jesu mreže koje koriste javnu telekomunikacionu infrastrukturu, kao što je Internet, kako bi obezbedili udaljenim lokacijama ili individualnim korisnicima siguran pristup mreži njihove organizacije. VPN mreže teže izbegavanju

iznajmljivanja namenskih skupih sistema prenosa i cilj im je obezbeđivanje iste ili skoro iste bezbednosti ali po mnogo nižoj ceni [1].

VPN enkapsulira podatke u posebne pakete i prenosi ih kroz više mrežnih uređaja kroz više raznih mreža čuvajući ih privatnim od ostalih uređaja na jednoj ili više lokalnih mreža ili *wide area network*. Postoji više različitih klasifikacija, implementacija i načina upotrebe VPN mreža.

Veliki broj tehnologija i protokola se koristi da omogući implementaciju *site-to-site* i *remote access* VPN mreža. Neke od tehnologija i protokola koji se koriste kod *site-to-site* VPN mreža jesu *IPsec*, *Generic Routing Encapsulation*, *Layer Two Tunneling Protocol*, *Draft Martini pseudowires*, *IEEE 802.1Q tunneling*, *MPLS Label Switched Paths*, dok se za *remote access* implementaciju koriste *Layer Two Forwarding Protocol*, *Point-to-Point Tunneling Protocol*, *Layer 2 Tunneling Protocol*, *IPsec*, *Secure Sockets Layer*.

Načini implementiranja VPN konekcije su različiti. Generalno se prilikom kreiranja VPN konekcije koriste neki inicijalizacioni parametri koji se nalaze u nekom fajlu ako se implementacija vrši na računaru. Često se u njemu nalazi putanja ka fajlu u kojem se nalaze ključevi, sertifikati, šifre, tj. neki tajni podaci. Ti podaci su nekada javni (kao što je javni ključ) ali veoma često, kada se ne koristi sertifikaciono telo, oni su tajni i od njihove tajnosti zavisi uspešnost celokupnog procesa šifrovane VPN veze.

Kada se VPN aplikacija koristi na strani kompanije sa računara koji je fizički u prostorijama kompanije ne postoji tako velika bojazan za bezbednost fajla sa tajnim podacima. Problem nastaje kada se konekcija ostvaruje sa javnih mesta kao što su Internet kafei, aerodromi i slično ili kada se laptop iznosi izvan prostorija kompanije. Tada je sadržaj fajla koji omogućava otključavanje VPN konekcije mnogo ranjiviji. Posebno ako se on mora preneti na fajl sistem nebezbednog računara. Postoje razni scenariji koji su realni i mogu dovesti do toga da zlonamerna osoba dođe do sadržaja tog fajla.

Idealno rešenje za navedeni problemi jeste primena smart kartica. Kartica je izvan fajl sistema, pristup podacima je zaštićen PIN kodom, a podaci se mogu čuvati i razmenjivati u šifrovanom obliku. Ipak, i ovde postoje problemi u primeni. Čuvanje podataka na kartici zahteva njihovo čitanje od strane VPN aplikacije koja uspostavlja komunikaciju. Postoji puno različitih VPN aplikacija te postoji problem dorade svake aplikacije ponaosob sa funkcionalnošću korišćenja smart kartice.

Sa druge strane, pošto aplikacije standardno znaju da rade sa fajlovima i generalno očekuju inicijalizaciju preko fajlova u fajl sistemu moguće je primeniti virtuelni fajl sistem. On se instalira u radnoj memoriji računara i ponaša se regularno kao element fajl sistema.

U ovom radu predloženo je moguće rešenje za čuvanje bezbednosno važnih parametara standardnih VPN aplikacija. Implementiran je virtuelni fajl sistem u memoriji *Linux/UNIX* sistema koji čita tajni sadržaj sa Java kartice i smešta ga u fajl koji sistem i korisnik mogu regularno da koriste. Po prestanku korišćenja fajl se briše iz memorije i ne ostavlja nikakav trag na memorijskim medijumima računarskog sistema.

2. PRIMENJENE TEHNOLOGIJE

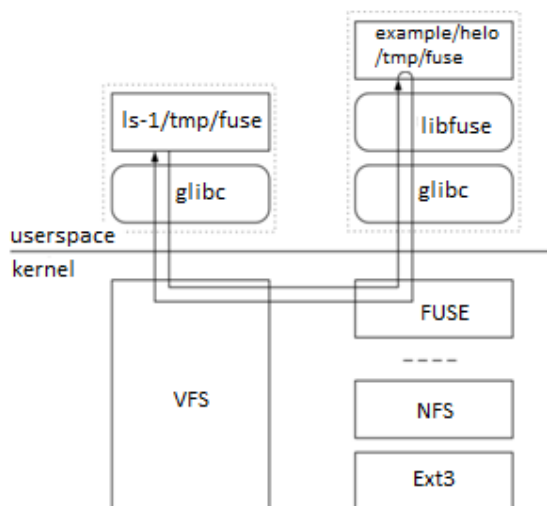
Za realizaciju predloženog rešenja potrebno je primeniti sledeće tehnologije.

FUSEAPI

Kao prvi korak rešenja potrebno je realizovati fajl sistem koji egzistira u memoriji računara, takozvani virtuelni fajl sistem (*VFS*) [2]. Kod *Linux/UNIX* sistema postoji više načina da se realizuje *VFS*. Jedan pristup se odnosi na izmenu kernela. Drugi pristup jeste realizacija modula za kernel koji bi se bavio obradom sisemskih poziva upućenih fajl sistemu. Treći pristup, kao što je *FUSE* [3], koristi već napravljene module.

Filesystem in Userspace jeste kernel modul za *Linux/UNIX* familiju operativnih sistema. On omogućava ne privilegovanom korisniku kreiranje sopstvenog fajl sistema bez izmene samog kernela. Realizovan je praktično u dve ravni. U korisničkom prostoru se izvršava sam kod korisničkog fajl sistema, dok se *FUSE* modul nalazi umetnut u kernel i praktično čini most između kernela i korisničkog fajl sistema.

FUSE je praktičan pri pisanju virtuelnih fajl sistema. Za razliku od tradicionalnih fajl sistema koji podatke skladište i čitaju na hard disku, virtualni fajl sistemi u stvari ne skladište podatke sami. Oni se više ponašaju kao pogled ili translacija nekog postojećeg fajl sistema ili uređaja za skladištenje. U principu, bilo koji resurs dostupan za *FUSE* implementaciju može biti iskorišćen u obliku fajl sistema. Slika 1 principijelno objašnjava funkcionisanje *FUSE* fajl sistema.



Slika 1. Principijalna šema funkcionisanja *FUSE*-a

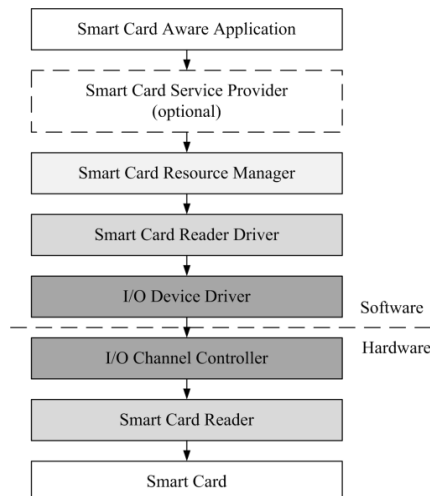
U korisničkom prostoru se izvršava aplikacija koju je korisnik napisao. Ona predstavlja nadogradnju fajl sistema sa funkcionalnostima koje su korisniku interesantne. Interfejs prema učitanoj *FUSE* modulu predstavlja *libFUSE* biblioteka koja se nalazi na vrhu *GNU*-ove *C* biblioteke poznate kao *glibc* [4]. Korisnik kroz korišćenje metoda, struktura i funkcija koje se nalaze u *libFUSE* biblioteci komunicira sa *FUSE* modulom.

FUSE modul se nalazi u kernelu i služi kao most između aplikacije u korisničkom prostoru i virtuelnog fajl sistema.

VFS je apstraktni nivo na vrhu konkretnog fajl sistema. Njegova namena je da unificira pristup aplikacija bilo kom fajl sistemu. Ponaša se kao interfejs kroz koji aplikacije pristupaju konkretnim fajl sistemima.

PC/SCliteAPI

Personal Computer / Smart Card jeste specifikacija kojom se standardizuju međusobne veze između personalnih računara i *ICC* uređaja. Specifikacija je razvijena sa ciljem da se na što jednostavniji način na personalnim računarima koriste smart kartice. Najveća prednost primene *PC/SC* tehnologije jeste činjenica da aplikacija koja je koristi ne mora da vodi računa o drajverima čitača smart kartice prilikom komunikacije sa smart karticom. Šta više, aplikacija može koristiti bilo koji čitač smart kartica koji je kompatibilan sa *PC/SC* standardom [4]. Struktura *PC/SC* arhitekture je prikazana na slici 2.



Slika 2. PC/SC arhitektura

PC/SC arhitektura je višeslojna i sastavljena je iz osnovnih i opcionalnih elemenata:

- *Smart Card Service Provider* - iako je opcional, on može biti veoma koristan jer pruža prijateljski standardizovani interfejs prema smart kartici. Uglavnom ga proizvode sami proizvođači smart kartica.
- *Smart Card Resource Manager* - je srce PC/SC arhitekture. Njegova strategijska pozicija koju zauzima prilikom toka transakcija između aplikacija i smart kartica omogućava simultani pristup datim čitačima kartica. On koristi svoja saznanja kako bi pomogao aplikacijama da identifikuju ili lociraju smart karticu. Rezultat ovakvog pristupa je jednostavniji drajver i lakši pristup kartici. Proizvode ga i razvijaju proizvođači operativnih sistema.

PC/SC je razvijen za Windows OS gde je integrisan u sam OS kroz *winscard.dll*. Zahvaljujući *open-source* zajednici dostupan je i za UNIX, GNU/Linux i Mac OS kroz PC/SCLite projekat na alioth.debian.org.

Sam API je pisan u ANSIC-u tako da se može koristiti na većini kompajlera. On ne koristi kompleksne i velike strukture kao što su vektori i slično. CAPI emulira *winscardAPI* koji se koristi na Windows platformi. On je smešten u biblioteku koja se zove *libpcsc-lite.so* koja se povezuje na aplikaciju.

Java Card Standard

Java Card je tehnologija kojom se omogućava korišćenje Java tehnologija na smart karticama i drugim uređajima sa ograničenim hardverskim resursima. Opisana je kroz Java Card specifikaciju i sa programerske strane realizuje se primenom Java CardAPI-ja.

Java Card Application Environment je licencirano na OEM osnovi kod proizvođača smart kartica i zastupljeno je na više od 90 procenata svih proizvedenih smart kartica širom sveta. Sa obzirom na ograničene memorijske resurse i procesorsku snagu Java kartice Java Card Specification je znatno redukovana u odnosu na Java Language Specification. To praktično znači da znatnu funkcionalnost Java Card ne podržava. Java Card tehnologija ima neke prednosti.

- Nazvisnost od platforme - Apleti napisani u Java Card tehnologiji koji su kompatibilni sa Java CardAPI specifikacijom mogu raditi na bilo kojoj Java kartici.

- Mogućnost pokretanja više aplikacija - *Java* programski jezik omogućava sigurno pokretanje više apleta na jednoj kartici.
- Naknadno instaliranje aplikacija - Nakon izdavanja kartice krajnjem korisniku moguće je naknadno instalirati nove aplete shodno korisnikovim potrebama.
- Fleksibilnost - *Java* je objektno orijentisani programski jezik i *Java Card* tehnologija nosi sa sobom sve prednosti koje pružaju OO jezici.
- Kompatibilnost sa postojećim standardima koji opisuju smart kartice - *Java CardAPI* je kompatibilan sa formalnim međunarodnim standardima kao što je ISO7816, i industrijskim standardima kao što je *Europay/Master Card/Visa*.

3. PREDLOŽENO REŠENJE

Aplikacija je realizovana kao servis u *ANSI C*-u [5] korišćenjem razvojnog okruženja *NetBeans IDE* verzija 6.9 [6]. Debugovanje aplikacije je rađeno pomoću *GDB* i *DDD* alata [7]. Ima tri source fajla i tri header fajla.

Po samom startovanju, aplikacija kreira jednu nit koja pravi zaseban servis. Ovaj servis je nazvan obradiKarticu.

Kao što mu ime kaže, njegov zadatak je da prvo inicijalizuje konekciju sa čitačem kartica. U zavisnosti od prisutnosti kartice u čitaču i postojanja apleta koji nosi fajl u sebi ova nit upisuje u zaseban memorijski prostor odgovarajući tekst. Po završetku nit prati promenu nastalu na čitaču i unutar apleta te shodno tome ponovo upisuje odgovarajući tekst u memorijski prostor. Očigledno je da ova nit radi sa *PC/SCLite* bibliotekom koja je u našem slučaju bila verzija 1.5.3. Aplikacija koristi većinu funkcija i struktura *PC/SCLite* biblioteke.

Kada se napravi nit obradiKarticu aplikacija dalje pravi fajl sistem u memoriji. Ovaj deo aplikacije koristi *FUSEAPI* koji je u našem slučaju bio verzija 2.8.1.

Fajl sistem koji se realizuje sastoji se od praktično samo jednog fajla koji nosi sa sobom *preshared* ključeve potrebne za ostvarivanje *VPN* konekcije. Fajl ne sme da se edituje niti menja i može da ga vidi i koristi samo aplikacija zadužena za pravljenje *VPN*-a.

Polazna struktura kod pravljenja *FUSE* fajl sistema jeste *FUSE_lowlevel_ops*. U toj strukturi se definišu pokazivači prema funkcijama koje će "presretati" uobičajene akcije sa fajlovima i obrađivati ih na način kako korisnik želi:

```
static struct FUSE_lowlevel_ops olja_ll_oper = {  
    .lookup = olja_ll_lookup,  
    .getattr = olja_ll_getattr,  
    .readdir = olja_ll_readdir,  
    .open = olja_ll_open,  
    .read = olja_ll_read}
```

U našem slučaju implementirane su četiri funkcionalnosti: *getattr*, *readdir*, *open* i *read*.

Prototip funkcije *getattr* je *getattr(const char* path, struct stat* stbuf)*. *Getattr* funkcija uzima parametre atributa fajla na koji pokazuje rukovalac. Funkcija popunjava elemente *stat* strukture parametrima fajla. Ukoliko je potrebno naknadno dorađivanje tih parametara to se može uraditi u okviru ove funkcije.

Prototip funkcije *readdir* je *readdir(const char* path, void* buf, FUSE_fill_dir_t filler, off_t offset, struct FUSE_file_info* fi)*. Funkcija vraća jednu ili više posebnih struktura zvanih *dirent*. Ovo je najkompleksnija *FUSE* funkcija. Zadatak joj je da za datu putanju vrati sadržaj

direktorijuma. Sposobna je da vraća i brojeve grešaka ukoliko se desi greška pri pravljenju strukture direktorijuma.

Prototip funkcije *open* je *open(const char* path, struct FUSE_file_info* fi)*. Funkcija otvara dati fajl. Ukoliko joj se prenese samo ime fajla ona proverava da li fajl postoji.

Prototip funkcije *read* je *read(const char* path, char *buf, size_t size, off_t offset, struct FUSE_file_info* fi)*. Funkcija čita *sizebytes* broj bajtova u *bufferbuf*. Kao odziv vraća broj prenetih bajtova ili 0 ukoliko je *offset* bio veći nego što je stvarna dužina fajla.

Ovaj deo aplikacije čita sadržaj memorije koji je prethodno spremila nit obradiKarticu i plasira ga kao običan fajl kroz *read* poziv. Ostale funkcionalnosti su neophodne kako bi se fajl sistem mogao *mount*-ovati i kako bi mu se moglo prići.

Pored programa koji je napisan za računar potrebno je i napraviti aplet koji se nalazi na *Java* kartici. Taj aplet nosi sa sobom sadržaj fajla u obliku niza i par funkcija potrebnih za funkcionisanje apleta.

Funkcija kojom se na osnovu *AID*-a odabira željeni aplet jeste funkcija *select*. U nju je ugrađena provera broja netačnih pokušaja unosa *PIN*-a. Ukoliko je broj pokušaja netačnog unosa *PIN*-a veći tada je aplet blokiran. Takođe, postoje i funkcije koje verifikuju, čuvaju, resetuju i čitaju *PIN*.

Ukoliko je unet pravilan *PIN* tada se može koristiti funkcija koja šalje sadržaj niza koji čuva fajl i na taj način ga predaje aplikaciji koja se nalazi na računaru kako bi ga ona ponudila kroz virtuelni fajl sistem.

U aplikaciji su definisane vrednosti statičkih promenljivih kojima se dogovorno, kroz *APDU* naredbe, pozivaju određene funkcije apleta i kojima aplet obaveštava aplikaciju o njegovom statusu.

4. ZAKLJUČAK

U radu je predložen pouzdaniji način za čuvanje parametara *VPN* veza i izvršena je realizacija i testiranje predloženog rešenja. Poboljšanje se odnosi na sigurnije čuvanje i korišćenje bezbednosno osetljivih parametara i na njihovo izmeštanje iz običnog fajla na *Java* karticu.

Današnje smart kartice imaju velike mogućnosti. U radu je korišćena samo funkcionalnost čuvanja podataka i pristup preko *PIN* broja i odgovarajuće *APDU* komande. U budućem radu moguće je iskoristiti i dodatne funkcionalnosti kartica što bi učinilo još sigurniju zaštitu datih ključeva.

U radu je pokazano da se može implementirati virtuelni fajl sistem na *Linux/UNIX* sistemima korišćenjem dostupnog, *GPL* licenciranog, *FUSEAPI*-ja. Iako postoji prisutan manjak dokumentacije i odgovarajuće literature koja objašnjava njegovo korišćenje, *FUSEAPI* se pokazao kao veoma efikasno i elegantno rešenje.

Takođe je pokazano da je moguće izvesti sopstvenu realizaciju interfejsa prema smart kartici na *Linux/UNIX* sistemima korišćenjem standardnog *PC/SCLiteAPI*-ja i na *Java* kartici realizovati celokupnu aplikaciju potrebnu za skladištenje ključeva.

Predloženo rešenje je transparentno u smislu upotrebe u različitim scenarijima korišćenja. Transparentnost leži u realizaciji rešenja kroz "običan" fajl sistem koji zna da koristi svaka potencijalna aplikacija.

Ovo rešenje je potrebno ispitati u realnoj produkciji u smislu brzine i praktičnosti upotrebe. Bilo bi interesantno doraditi rešenje kroz grafički interfejs koji bi ga učinio više korisnički orijentisanim. Postoji potreba da se realizuje "punilac" fajla koji bi jednostavno preneo taj fajl na *Java* karticu.

Naravno, ograničenje koje ima kartica u memorijskom smislu je i dalje prisutno i može se rešiti jedino upotrebom specijalizovanih kartica (npr. laserske kartice) koje imaju veliku količinu memorije.

LITERATURA

- [1] http://en.wikipedia.org/wiki/Virtual_private_network
- [2] Love Robert, Linux Kernel Development, Third Edition, Pearson Education, Inc., Indiana, 2010.
- [3] http://en.wikipedia.org/wiki/Filesystem_in_Userspace
- [4] http://en.wikipedia.org/wiki/GNU_C_Library
- [5] <http://www.gemalto.com/techno/pcsc/other/aboutpcsc.html>
- [6] King K. N., C Programming, A Modern Approach, Second Edition, W.W. Norton, London, 2008.
- [7] Petri Jurgen, NetBeans Platform 6.9, Developer's Guide, Packt Publising Ltd, Birmingham, 2010.
- [8] Matloff N. and P. J. Salzman, The Art of Debugging with GDB, DDD and Eclipse, No Starch Press, San Francisko, 2008.