SPRAWOZDANIE Z PROJEKTU

Autor: Mieszko Makowski
Przedmiot: Programowanie sieciowe wspierające aplikacje bezpieczne
Temat: „SafeXfer" – bezpieczna wymiana plików w architekturze klient-serwer (TCP/SCTP)
Środowisko: Kali Linux (x86_64), GCC, glibc, OpenSSL

1.Streszczenie

Celem projektu było zbudowanie prototypu bezpiecznej wymiany plików SafeXfer w architekturze klient–serwer. Rozwiązanie udostępnia auto-odkrywanie serwera (UDP multicast), sesję TCP z logowaniem (PBKDF2-HMAC-SHA256), interaktywnego klienta oraz operacje na plikach: ls, rm <plik>, get <plik>, put <plik>. Zaimplementowano własny, prosty protokół TLV do sterowania i transferu danych w jednym połączeniu TCP.

Wersja obecna spełnia główne założenia funkcjonalne; w sekcji „Prace dalsze" wskazano rozszerzenia: tryb demona + syslog, równoległość, TLS oraz SCTP multi-stream. Założenia i wymagania pochodzą z konspektu projektu (cel, moduły, porty/protokoły).

2. Cel i zakres

Zgodnie z konspektem, system powinien umożliwiać: interaktywny transfer (get, put, ls, rm), serwer współbieżny (demon + syslog), kanał TCP do sterowania/pobierania pojedynczego pliku, SCTP do wielu plików równolegle oraz UDP multicast 224.0.0.251:54321 do auto-discovery.

Wersja dostarczona (MVP):

✅ UDP multicast NetDiscovery (224.0.0.251:54321) – wykrywanie serwera.

✅ TCP SrvCore (port 2121) – logowanie + sesja interaktywna.

✅ AutoGuard – loginy/hasła PBKDF2-HMAC-SHA256 (100k iteracji).

✅ Operacje plikowe (FileEngine w SrvCore): ls, rm, get, put.

✅ Protokół TLV (sterowanie i dane w jednym TCP).

⏳ Planowane: demonizacja + syslog, równoległość, SCTP 9899 (multi-stream), ewentualnie podział kanałów sterowanie/transfer „PORT/PASV-like".

3. Architektura i moduły

```
/safexfer
├─ server/
│   ├─ main.c                  (uruchamia NetDiscovery w wątku + start_tcp_server)
│  ├─ srvcore.c/.h     (pętla TCP, sesje, FileEngine: ls/rm/get/put)
│  ├─ autoguard.c/.h   (PBKDF2, weryfikacja kont)
│  ├─ netdiscovery.c   (UDP multicast listener)
│  ├─ accounts.txt     (login:salt_hex:pbkdf2_hex)
│  └─ storage/         (repozytorium plików)
├─ client/
│  ├─ main.c           (discovery → run_client(IP))
│  ├─ cli.c            (logowanie, pętla komend, get/put)
│  └─ netdiscovery.c   (UDP multicast probe)
├─ common/
│  ├─ tlv.c/.h         (ramki TLV, readn/writen)
└─ Makefile
```

Role modułów (wg konspektu: SrvCore, FileEngine, AutoGuard, NetDiscovery, CLI-Client) – odwzorowane w strukturze projektu.

4. Protokół i porty

4.1. Auto-discovery (UDP multicast)

- Grupa: **224.0.0.251**, port **54321**.
- Klient wysyła „DISCOVER_SAFEXFER" → serwer odpowiada unicastem „SAFEXFER_SERVER" (z IP).
- Zgodnie z konspektem: „UDP multicast 224.0.0.251:54321 – auto-discovery".

4.2. Kanał sterowania/transferu (TCP 2121)

- Po wykryciu IP, klient łączy się z serwerem TCP **2121**.
- Logowanie TLV → po sukcesie pętla komend (`ls`, `rm`, `get`, `put`).
- W aktualnej wersji **transfer plików** również po tym samym TCP (upraszcza MVP).
- W konspekcie przewidziano dodatkowo: „TCP – sterowanie + pojedynczy plik"

oraz **SCTP 9899** dla wielostrumieniowego transferu – sekcja *Prace dalsze*.

4.3. TLV (Type–Length–Value)

Nagłówek: 1 bajt `type`, 2 bajty `length` (big-endian), następnie `value[length]`.

Typy sterujące:

- `0x01` LOGIN, `0x02` PASSWORD
- `0x10` OK, `0x11` ERROR
- `0x20` CMD (tekst: `ls`, `rm X`, `get X`, `put X`), `0x21` TEXT (odpowiedź)

Typy transferu pliku:

- GET (serwer→klient): `0x31 FILE_INFO` (8 B rozmiar, BE), `0x32 FILE_CHUNK`, `0x33 FILE_END`
- PUT (klient→serwer): `0x51 PUT_CHUNK`, `0x52 PUT_END`
  Rozmiar ramki danych (wartość TLV): domyślnie **4096 B**; łatwo zwiększyć.

5. Implementacja

- **Język:** C (gniazda BSD, `pthread` dla wątku discovery).
- **Kompilacja:** `make` (Makefile kompiluje serwer i klienta; serwer linkuje `-lssl -lcrypto -pthread`).
- **Autoryzacja (AutoGuard):** `PKCS5_PBKDF2_HMAC(…, iter=100000, SHA-256)`.
- **Format kont:** `login:salt_hex:pbkdf2_hex` (np. dla `admin:test123`, sól `112233`):
  `admin:112233:26c8ff17c3559cefb31c9599f9eeb8e0e6177a9599f0218aac0cef9d036b1142`
- **Repozytorium plików:** `server/storage/` (tworzone automatycznie).
- **Twarde zabezpieczenia ścieżek:** odrzucenie nazw z `..`, `/`, `\` (ochrona przed path traversal).
- **I/O:** transfer w kawałkach (CHUNK) z prostą sygnalizacją końca (FILE_END / PUT_END).

---

6. Instrukcja budowania i uruchomienia

6.1. Wymagania (Kali)

```
build-essential gdb valgrind pkg-config make libsctp-dev lksctp-tools libssl-dev
```

6.2. Budowanie

```
make clean && make
```

6.3. Uruchomienie

Serwer (Terminal 1):

```
./server/server
# Discovery: 224.0.0.251:54321
# TCP: nasłuch na 2121
```

Klient (Terminal 2):

```
./client/client
# Po discovery: Login/Hasło → sesja SafeXfer>
```

7. Przykładowe sesje testowe

Logowanie + ls / put / get / rm / exit

```
$ ./client/client
Znaleziono serwer SafeXfer pod IP: 192.168.1.18
Login: admin
Hasło: test123
✅ Zalogowano! Komendy: ls | rm <plik> | get <plik> | put <plik> | exit
SafeXfer> ls
(empty)
SafeXfer> put raport.txt
OK
SafeXfer> ls
raport.txt   1024 bytes
SafeXfer> get raport.txt
```

```
Pobrano raport.txt (1024/1024 bajtów)
SafeXfer> rm raport.txt
OK
SafeXfer> exit
```

Kontrola portu TCP:

```
nc -zv <IP_serwera> 2121
```

8. Testy, wyniki i obserwacje

- **Discovery** działa w sieci lokalnej (TTL=1). Na pojedynczym hoście (loopback) zalecane włączenie `IP_MULTICAST_LOOP=1`.
- **Autoryzacja**: poprawne logowanie dla rekordów z `accounts.txt`; błędne hasło → `TLV ERROR`.
- **Transfer**: pliki do kilkuset MB możliwe; przepustowość zależy od `TLV_MAX_VALUE` (domyślnie 4096 B).
- **Odporność**: sanityzacja nazw plików; proste komunikaty błędów.
- **Stabilność**: serwer obsługuje sesje sekwencyjnie (MVP). Dla wielu klientów należy dodać forking/wątki/asynchroniczność.

9. Aspekty bezpieczeństwa

Zaimplementowane:

- PBKDF2-HMAC-SHA256 (100k) + sól HEX per użytkownik.
- Blokada path traversal (brak `..`, `/`, `\` w nazwach).
- Protokół binarny TLV (prostota parsowania, brak evalowania tekstu).

Ryzyka/ograniczenia (MVP):

- Brak **TLS** – hasła i dane idą jawnie w LAN; do użycia w sieci zaufanej/na labach.
- Brak rate-limit/lockout – możliwe brute force offline/online.
- Jednowątkowość – blokada przy długich transferach.

Rekomendacje (prace dalsze):

- TLS (OpenSSL) dla kanału TCP; HSTS po stronie HTTP-UI (jeśli kiedyś).
- Limit prób logowania, dziennik audytowy (syslog).
- Demonizacja i logowanie do `syslog` (wymóg kursowy).
- Równoległość (fork/pthreads/epoll).
- **SCTP 9899** – multi-stream dla równoległych plików (zgodnie z konspektem).
- Sumy kontrolne plików (SHA-256) po transferze.

10. Wnioski

Zbudowano funkcjonalny prototyp **SafeXfer** zgodny z założeniami funkcjonalnymi konspektu: discovery, logowanie PBKDF2, sesja interaktywna, operacje plikowe i transfer w oparciu o TLV. Projekt jest gotowy do rozszerzenia o elementy wymagane w pełnej wersji (demon + syslog, współbieżność, TLS, SCTP). Dzięki prostemu TLV i modularnej strukturze kodu, dodanie kolejnych komend i mechanizmów (np. integrytet, rate-limit) jest nieskomplikowane. Założenia modułowe i porty/protokoły odzwierciedlono zgodnie z konspektem.