VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta informačních technologií

ISA - PROJEKT

isa-ldapserver: server pro LDAP komunikaci

2023/2024

Xpejch08 - Štěpán Pejchar

Úvod

Zadání projektu

Cílem projektu bylo vytvořit server, který slouží pro LDAP komunikaci s LDAP klientem. Implementován je v jazyce C/C++. Server je nabindovaný jak na IPv4 tak IPv6 a je paralelní a neblokující. Server nevyžaduje autentizaci a nepodporuje mezinárodní znaky. Server umí rozpoznat pouze bind request, search request a unbind request, ostatní zprávy ignoruje. Podporované filtry jsou:

- And
- Or
- Not
- Equality Match
- Substrings

Server by měl pomocí zadaných filtrů vyhledat záznamy v databázi, která má formát csv souboru obsahující 3 položky a to sice:

- Jméno cn
- Login uid
- Email mail

Server vrátí klientovi search result entry pro každý odpovídající záznam a akonec klientovi odešle zprávu search result done. Klient v tu chvíli ví, že všechny záznamy byly přijaty. Server poté čeká na unbind request zrávu od klienta, čímž uzavírá komunikaci se serverem.

Teorie LDAP komunikace

LDAP protokol neboli Lightweight Directory Access Protocol je protokol, který slouží pro ukládání a přístup k datům na adresářovém serveru. Jednotlivé položky by měli být ukládány ve stromové struktuře, to v naší odlehčené verzi neplatí. Jedno z nejčastějších použití LDAP protokolu je autentizace userů. V nějaké databázi máme uložená uid - unikátní username a hesla. Pomocí LDAP protokolu můžeme ověřovat zda uživatel tyto údaje zadal správně. LDAP protokol může být používán různými aplikacemi jako je například Docker nebo Jenkins. [1]

LDAP formátování a konkrétní podporované zprávy

LDAP zpráva je formátována ve stylu ASN.1 basic encoding rules (BER). Formátování LDAP zprávy je case insensitive tudíž nezáleží na velikosti písmen. Konkrétní zprávy jsou popsány v dokumentaci RFC. Jednotlivé byty, které slouží pro posílání zprávy, jsou zákodované v hexadecimální soustavě. V naší aplikaci není podporován simple bind, tudíž klient se nemusí k serveru nejprve přihlašovat. [2] [3] [7]

Bind Request

Bind request je zpráva, kterou odesílá klient serveru. Jejím účelem je navázání spojení se serverem. V mojí implementaci bind request neobsahuje ověřovací údaje klienta.

Bind Response

Bind response je zpráva, kterou server odesílá potom, co dostal správně formátovanou zprávu bind request od klienta.

Search Request

Search requestem klient zasílá serveru data o konkrétním filtru, kterým má server data vyfiltrovat. Data zahrnují například base, scope, size limit a také konkrétní filtr. Díky této správě server může vyhledávat specifické údaje v databázi.

SearchResEntry a SearchResDone

Zpráva searchResEntry posílá klientovi konkrétní vyfiltrovaná data. Správa searchResDone informuje klienta, že žádná další data server nepošle.

Typy podporovaných filtrů

Filtry jsou v LDAP komunikaci využívýny pro vyhledávání a selekci konkrétních dat. Pomocí těchto filtrů můžeme specifikovat konkrétní kritéria na základě kterých potom vyhledáváme v databázi. Filtry mohou být různě nakombinovány a zanořovány do sebe.

Equality Filtr

Equality filtr filtruje data na základě porovnání s konkrétní hodnotou. Pokud je hodnota stejná, filtr vrátí searResEntry. Například pokud máme filtr (cn=Stepan Pejchar), server by vrátil záznam s uživatelem Stepan Pejchar, pokud by takový záznam v databázi byl. Pokud bychom například zadali filtr (cn=Stepan Pej) filtr by žádný záznam nevrátil, jelikož takový záznam v naší databázi neexistuje.

Substring Filtr

Substring filtr filtruje data na základě podřetězců a používá se pomocí znaku *. Máme typy, jakými můžeme substring filter použít. První typ je filtr s hvězdičkou na začátku například (cn=*Stepan). Tento filtr je kombinace equality filtru a substring filteru, kde říkáme, že chceme najít všechna cn, kde křestní jméno je štěpán a řetězec nacházející se před křestním jménem může být libovolý. Filtr tedy najde všechny Stepany. Druhý způsob použití je hvězdička na konci například (cn=Pejchar*). Tímto filtrem říkáme, že data musí obsahovat substring Pejchar, ale cokoli je za ním je jedno. Tento filtr vrátí jeden záznam Stepan Pejchar z databáze. Poslední typ filtru je použití dvou hvězdiček zároveň. Tím říkáme, že chceme aby data obsahovali specifický substring, ale to co je před ním nebo za ním nás nezajímá. Filtr může vypadat například takto (cn=*Ste*). Tento filtr by vrátil všechny záznamy, které kdekoli v cn obsahují podřetězec Ste. [8]

And Filtr

And filtr nám pomáhá kombinovat více různých filtrů dohromady pomoci logické funkce. Pokud kombinujeme například 2 filtry zároveň pomocí & může to vypadat například takto (&(cn=*stepan)(cn=p*)). Tímto filtrem říkáme, že chceme vyhledat všechny záznamy, jež

křestní jméno je štěpán a zároveň jejich přijmení začíá na P. Tento filtr v naší databázi vrátí pouze záznam Stepan Pejchar.

Or Filtr

Stejně tak jako And filtr můžeme použít i Or filtr, který kombinuje filtry pomocí logické funkce nebo. Filtr by tedy mohl vypadat takto (|(cn=*stepan)(cn=p*)). Tento Filtr by vrátil všechny záznamy z databáze kde se useři buď jmenují křestním jménem Stepan, nebo jeji přijmení začíná na p.

Not Filtr

Not filtr slouží pro negaci konkrétního filtru. Používá se pomocí vykřičníku například takto (!(cn=Pejchar*)). Tento Filtr vrací všechny záznamy kromě těch, jejichž přijmení je pejchar.

Implementace

ldap.cpp a ldap.h

Soubory slouží pro základní implementaci serveru. LDAP protokol běží pomocí TCP komunikace, tudíž v základním smyslu je LDAP server TCP server. V hlavičkovém souboru definuji strukturu Args, pro správu argumentů. Obsahuje cestu k souboru s databází a také hodnotu portu, která je implicitně nastavena na 389. Následně jsem naimplementoval typický TCP server Ve funkci bindSocket nastavuji optname na SO_REUSEADDR. Díky této možnosti se můžu při nesprávném vypnutí programu opět připojit na stejný port bez čekání na ukončení timeout. Tato možnost byla použita hlavně při implementaci, ve finální verzi se při signálu SIGINT všechny sokety uvolní jak je definováno ve funkci closeSocket a voláno v signal_callback_handler. Server nejprve vytvoří soket, následně mu přiřadí konkrétní adresu. Jako další následuje bindování soketu a poté server čeká na paket pomocí funkce listen. V této funkci také parsuji databázi do množiny fileContent. Z databáze odstraním whitespaces a uložím všechny záznamy zvlášť. Po přijetí paketu následuje funkce readPacket, která konkrétní paket přijímá. Po úspěšném přijetí se pomocí funkce fork vytvoří nový proces aby byl server paralelní a neblokující, jak je popsáno v zadání. Ve funkci readPacket následně volám funkci ldapProtocol, která spustí veškerou logiku LDAP komunikace.

ldapParser.cpp

Tento soubor implemetuje veškerou logiku ldap komunikace. Implementoval jsem ho pomoci znalostí získaných z RFC, informací o ldap protokolu, a pomoci programu wireshark, kde jsem odchytával ldap komunikaci. Logika je navržena podobně, jako lexikální analýza. Postupně volám pomocnou funkci getNextChar a kontroluji jestli byte, který je po zavolání uložen v proměnné messageByte odpovídá bytu, který by měl v danou chvíli přijít. Soubor tedy obsahuje 3 funkce, které tuto logiku implementují, a to sice funkci parseBegin, funkci bindReq a searchReq. Tyto funkce postupně procházejí obdrženou zprávu byte po bytu, přiřazují specifické informace, které konkrétní byty nesou, a volají funkce, které Implementují filtraci. [4] [6]

Funkce bindRes

V této funkci sestavuji a odesílám zprávu bind response klientovi. Funkce je volána hned po přečtení zprávy bind request. Vnitřní část zprávy definuji konstantně. Následně voláním pomocných funkci genereateMessageWithLen a generateMessageWithID sestavím zprávu, a odešlu ji zpět klientovi pomocí funkce write.

Funkce searchResultGen

Tato funkce má za úkol generovat zprávu searchResEntry. Funguje podobně jako funkce bindRes. Zprávu však generuje pomoci proměnné out ve které máme uložené vyfiltrovaná data. Do vrácené zprávy jsem mimo cn jména a emailu zadaného v zadání přidal i uid a dn zvlášť vrácená zpráva tedy vypadá například takto:
xpejch08
dn: uid=xpejch08
cn: Pejchar Stepan
uid: xpejch08
mail: xpejch08@stud.fit.vutbr.cz

Funkce searchResDone

Funkce searchResDone má za úkol poslat klientovi zprávu SearchResDone. Je implementována podobně jako funkce bindRes.

Funkce LDAPFilterGetter

Tahle funkce inicializuje a filter a volá příslušnou funkci pro handling specifického filtru. V jednom případě volá funkci pro spracování substring a equality filtru a v druhém případě volá funkci pro spracování AND, OR a NOT. Tato funkce je volána před funkcí filterLogic. Jejím úkolem je uložit data o konkrétních filtrech do struktury LDAPFilter, která je definována takto:

```
class LDAPFilter{
public:
   int lenOfFilter = 0, attributeIndex, filterType = -1;
   std::string attrValue;
   std::string attribute;
   std::vector <LDAPFilter> nestedFilters;
   std::map<std::string, int> attributeConst =
                        {{"cn", 0}, {"commonname", 0},
                        {"uid", 1}, {"userid", 1},
                        {"mail", 2}};
};
```

Každý filtr si ukládá pole dalších vnořených filtrů, dále si uchovává hodnotu konkrétního atributu (např. “Stepan Pejchar”), specifický atribut podle kterého vyhledáváme (např “cn”), a mapu, ve které bindujeme atributy na čísla, aby se s nimi lépe pracovalo. V mapě také definujeme, že například hodnota uid a hodnota userid je identifikována stejně.

Funkce filterLogic

Ve funkci filterLogic implementuji co jednotlivé filtry ve skutečnosti dělají. Funkce je volána rekurzivně dokud se neprovedou všechny vnořené filtry uložené v poli nestedFIlters. Funkce vždy nejprve vykonstruuje regex pattern podle daného filtru a následně pomocí funkce regex_search vyhledá výsledné hodnoty v poli filter attributes, kde jsou uložené naparsované vstupy z databáze.

decodeLdap.cpp a decodeLdap.h

V hlavičkovém souboru decodeLdap.h implementuji veškeré třídy potřebné pro správný chod programu.

LDAPMessage

Třída obsahující generickou ldap zprávu. Obsahuje její id, délku, typeOfMessage (např. BINDREQ) a reálnou zprávu v proměnné message.

decodeLdapFunctions

Třída definující veškeré funkce popsané výše. Dále se v ní ukládají LDAPMessage, jednotlivé data z databáze, výstup pro generování searchResEntry, aktuálně zpracovávaný byte, odkaz na paket pro komunikaci, aktuální pozice ve zprávě, a struktura LDAPFilter popsaná výše.

Konstanty

V hlavičkovém souboru decodeLdap.h definuji několik konstant abych omezil používání tzv. magic numbers. Definuji konstanty pro vyjádření typu filtru např. const int AND = 0xA0, nebo const int SEARCHRESENTRY = 0x65.

decodeLdap.cpp

V tomto souboru implementuji pomocné funkce aby se zvýšila přehlednost kódu. Funkce, které jsou zde implementovány používám zejména v souboru ldapParser, ve kterém implementuji logiku protokolu. Mezi funkce patří getIDOfMessage, getMessage, getLenOfMessage, getNextChar, generateMessageWithID a generateMessageWithLen.

Testování aplikace

Aplikaci jsem testoval na operačním systému Kubuntu 22.04 pomocí nástroje ldapsearch v terminálu. Spuštěný server běžel na mém počítači tedy na localhost. Model mého počítače je: Lenovo Lenovo Legion 5 15ARH05H. Příklady a ukázky jednotlivých testů můžeme vidět níže. Testy jsem implementoval do souboru test.sh který je součástí adresáře. Funguje na bázi porovnávání reálných outputů a expected outputů definovaných ve složce expectedOut. Pro použití nástroje ldapsearch jak je popsáno níže musí být nejdříve spuštěn implementovaný server například pomocí posloupnosti příkazů: make clean, make, sudo ./isa-ldapserver -p 389 -f normal.csv. Server musí být spouštěn s root privileges. Testovací skript může být spuštěn buď pomocí “bash test.sh” nebo po upravení práv pro daný soubor příkazem “chmod +x ldap_test.sh” můžete použít ./test.sh. Ukázkový výstup například pro filtr `(cn=Pejchar*)` vypadá takto:

```
# extended LDIF
#
# LDAPv3
# base <> (default) with scope subtree
# filter: (cn=Pejchar*)
# requesting: ALL
#

# xpejch08
dn: uid=xpejch08
cn: Pejchar Stepan
uid: xpejch08
mail: xpejch08@stud.fit.vutbr.cz

# search result
search: 2
result: 0 Success

# numResponses: 2
# numEntries: 1
```

Ostatní ukázkové výstupy vypadají podobně s tím rozdílem, že ukazují víc částí s konkrétně vyhledanými usery, (blok začínající # xpejch08) nebo neukazují žádné a jsou k vidění v adresáři expectedOut.
Výstup po spuštění test.sh skriptu:

```
Test Passed for query: (cn=Pejchar*)
Test Passed for query: (uid=xpejch08)
Test Passed for query: (&(cn=*stepan)(cn=p*))
Test Passed for query: (&(!(mail=*a*))(mail=*en*))
Test Passed for query: userid=aaaaaaaa08
Test Passed for query: (|(cn=*stepan)(cn=p*))
Test Passed for query: (&(&(cn=*stepan)(uid=xp*))(mail=*p*))
```

Test1 equality a substring

spuštění ldap komunikace:sudo `ldapsearch -x -H ldap://localhost -x "(cn=Pejchar*)"`
filtr: cn=Pejchar*
Filtr by měl najít všechny usery jejichž přijmení je pejchar. Expected output je v souboru expectedOut/expected1.ldif.

Test2 equality

spuštění ldap komunikace: `ldapsearch -x -H ldap://localhost -x "uid=xpejch08"`

filtr:uid=xpejch08
Filtr by měl najít všechny usery, jejichž uid je ekvivalentní s xpejch08. Expected output je v souboru expectedOut/expected2.ldif.

Test3 and filter

spuštění ldap komunikace: `ldapsearch -x -H ldap://localhost -x "(&(cn=*stepan)(cn=p*))"`
filtr: (&(cn=*stepan)(cn=p*))
Filtr by měl najít všechny usery, jejichž křestní jméno je stepan a zároveň jejichž přijmení začíná na p . Expected output je v souboru expectedOut/expected3.ldif.

Test4 and a not filter

spuštění ldap komunikace: `ldapsearch -x -H ldap://localhost -x "(&(!(mail=*a*))(mail=*en*))"`
filtr: `(&(!(mail=*a*))(mail=*en*))`
Filtr by měl najít všechny usery, jejichž email neobsahuje podřetězec “a” a zároveň obsahuje podřetězec “en”. Expected output je v souboru expectedOut/expected4.ldif.

Test5 test mapování uid a userid

spuštění ldap komunikace: `ldapsearch -x -H ldap://localhost -x "userid=aaaaaaaa08"`
filtr: `userid=aaaaaaaa08`
Filtr by měl najít všechny usery, jejichž uid je rovno “`aaaaaaaa08`”. Expected output je v souboru expectedOut/expected5.ldif.

Test6 or test

spuštění ldap komunikace: `ldapsearch -x -H ldap://localhost -x "(|(cn=*stepan)(cn=p*))"`
filtr: `(|(cn=*stepan)(cn=p*))`
Filtr by měl najít všechny usery, jejichž křestní jméno je “stepan”, nebo jejichž přijmení začíná na “p”. Expected output je v souboru expectedOut/expected6.ldif.

Test7 test všech atributů zároveň + vnořený and

spuštění ldap komunikace: `ldapsearch -x -H ldap://localhost -x "(&(&(cn=*stepan)(uid=xp*))(mail=*p*))"`
filtr: `(&(&(cn=*stepan)(uid=xp*))(mail=*p*))`
Filtr by měl najít všechny usery, jejichž křestní jméno je “stepan”, uid začíná na “xp” a zároveň email obsahuje písmeno “9”. Expected output je v souboru expectedOut/expected7.ldif.

Bibliografie

[1] ASN.1 Basic Encoding Rules. (n.d.). OSS Nokalva. https://www.oss.com/asn1/resources/asn1-made-simple/asn1-quick-reference/basic-encoding-rules.html

[2] ASN.1 JavaScript decoder. (n.d.). lapo luchini. https://lapo.it/asn1js/

[3] Ellingwood, J. (2015, May 29). *Understanding the LDAP Protocol, Data Hierarchy, and Entry Components*. DigitalOcean. https://www.digitalocean.com/community/tutorials/understanding-the-ldap-protocol-data-hierarchy-and-entry-components

[4] https://gitlab.com/wireshark/wiki-editors. (n.d.). *Lightweight Directory Access Protocol (LDAP)*. Wireshark Wiki. https://wiki.wireshark.org/LDAP

[5] LDAPv3 Wire Protocol Reference: The ASN.1 Basic Encoding Rules. (n.d.). LDAP.com. https://ldap.com/ldapv3-wire-protocol-reference-asn1-ber/

[6] RFC 4519: Lightweight Directory Access Protocol (LDAP): Schema for User Applications. (n.d.). RFC Editor. https://www.rfc-editor.org/rfc/rfc4519

[7] ASN.1 Basic Encoding Rules. (n.d.). OSS Nokalva. https://www.oss.com/asn1/resources/asn1-made-simple/asn1-quick-reference/basic-encoding-rules.html

[8] How to write LDAP search filters | Atlassian Support. (2023, July 20). Atlassian Documentation. https://confluence.atlassian.com/kb/how-to-write-ldap-search-filters-792496933.html