Digital Whisper Technologic Papers

בניית לקוח Telnet

מאת נועם אלום

הקדמה

Telnet הוא פרוטוקול פשוט מאוד, אני חושב שבניית לקוח Telnet קטן זה פרויקט מצויין כדי ללמוד יותר על תכנות פרוטוקולים. ובאשר לי, הייתי צריך להשתמש ב-Telnet בפרויקט אחר שאני עובד עליו, ואין באמת ספרייה ברורה שאיתה כדאי לעבוד עם הפרוטוקול הזה ב-CPP (כמו למשל libssh ל-SSH). אולי משום שהפרוטוקול באמת כזה פשוט. ולכן, החלטתי לבנות ספרייה header only ל-Telnet.

- קובץ Header only הוא קובץ הכללה שבו מוגדר גם ההגדרה וגם הביצוע של הקוד.
- יש ספרייה מאוד נחמדה ב-GitHub ושמה-telnetpp, אני בחרתי לכתוב ספרייה משל עצמי כדי פחות להסתמך על מקורות משניים.

קצת רקע על Telnet

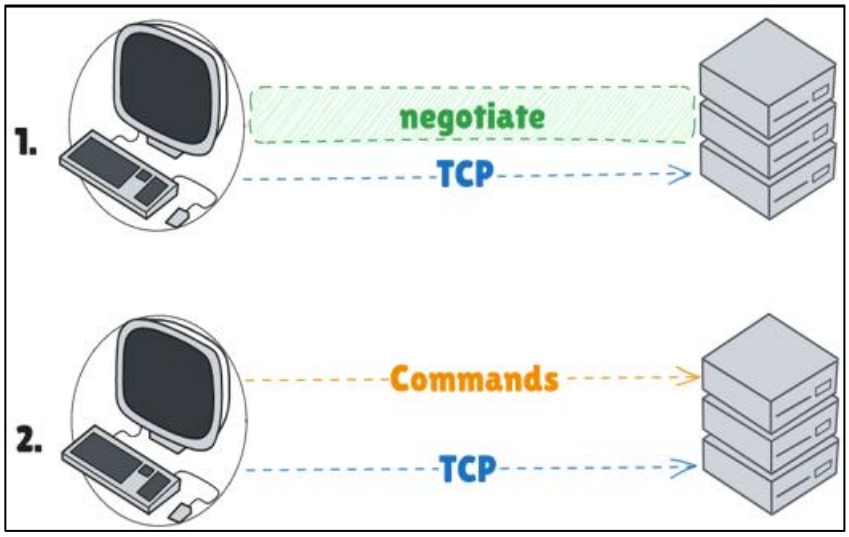
Telnet הוא פרוטוקול ותיק מאוד ותוכנן בתקופה שבה אבטחת מידע לא הייתה שיקול משמעותי בתכנון מערכות תקשורת. הוא פועל על גבי TCP ומשמש לתקשורת טקסטואלית אינטראקטיבית בין לקוח לשרת, לרוב באמצעות שורת פקודה.

אחת הבעיות המרכזיות ב-Telnet היא העובדה שכל הנתונים כולל שם המשתמש והסיסמה מועברים בטקסט גלוי ללא הצפנה כלשהי, זו בדיוק הסיבה לכך שהוא נחשב היום לפרוטוקול פגיע מאוד על אחת כמה וכמה כאשר מדובר ברשתות פתוחות או חשופות לאינטרנט.

למרות זאת, Telnet עדיין נמצא בשימוש באינספור מערכות, בעיקר בניהול ציודי תקשורת ישנים כמו נתבים, מתגים ומודמים, שבהם אין ממשק מודרני יותר (כמו SSH), או שבהם Telnet מופעל כברירת מחדל. בשטח, מתברר שגם פרוטוקול כל כך פשוט ולא מאובטח עדיין חי ובועט.

מחזור החיים של חיבור Telnet

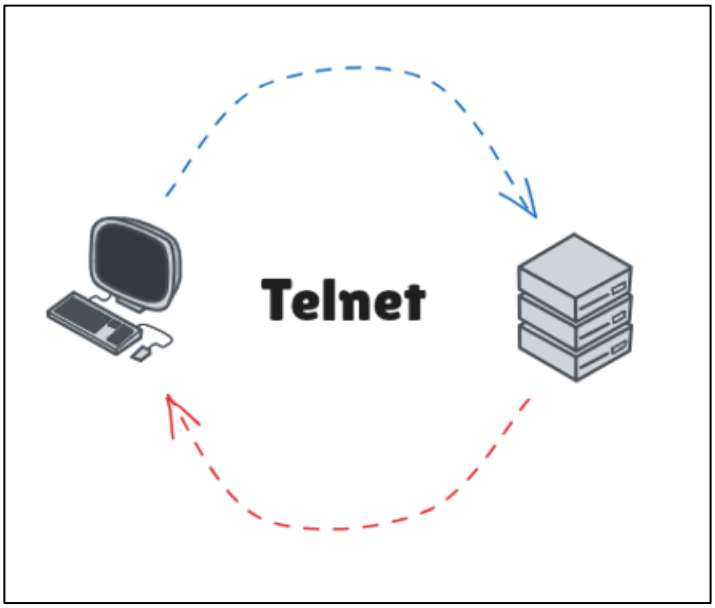
כדי לעזור לכם להבין את מחזור החיים של חיבור Telnet הכנתי את הדיאגרמה הבאה:



- ה-negotiation והפקודות על אותו חיבור TCP.

לאחר יצירת TCP Session על פורט מסוים (אמור להיות 23), שרת ה-Telnet אמור להתחיל את שלב ה-Negotiation, שלב זה מבטיח שהשרת והלקוח מסכימים על חוקים קבועים לאותו Session.

לאחר שלב ה-Negotiation, ניתן להתחיל לשלוח מחרוזות (strings), ורוב שרתי ה-Telnet אמורים להבין אותן כפקודות.



שלב ה-Negotiation

יחסי שרת/לקוח ב-Telnet מתוחזקים על ידי הפקודות הבאות: (לפי ה-RFC)

שם	קוד	משמעות
SE	240	סיום שלב תת-משא ומתן (Subnegotiation).
NOP	241	אין פעולה (No Operation).
Data Mark	242	סימון בזרם הנתונים במהלך סנכרון. מלווה בהתראה דחופה (Urgent) של TCP.
Break	243	שליחת תו BRK.
Interrupt Process	244	פקודת IP, הפסקת תהליך נוכחי.
Abort output	245	ביטול פלט.
Are You There	246	בדיקה אם הצד השני עדיין מחובר.
Erase character	247	מחיקת תו אחד.
Erase Line	248	מחיקת שורת קלט.
Erase Line	249	מתן רשות לשלוח נתונים.
SB	250	תחילת תת-משא ומתן עבור האפשרות שצוינה.
WILL (option code)	251	מציין רצון להתחיל או אישור שהצד מבצע את האפשרות שצוינה.
WON'T (option code)	252	סירוב לבצע או להמשיך לבצע את האפשרות שצוינה.
DO (option code)	253	בקשה או דרישה שהצד השני יבצע את האפשרות שצוינה.
DON'T (option code)	254	דרישה מהצד השני להפסיק לבצע את האפשרות שצוינה.
IAC	255	תו מיוחד המציין שפקודת Telnet עומדת להגיע (Interpret As Command).

בגיליון זה אני אתמקד בפקודות הבאות:

- IAC
- DO
- DONT
- WILL
- WONT

כמו שניתן לראות בטבלה, פקודות אלו משמשות בכדי להפעיל/לכבות אפשרויות בשרת או לקוח, אפשרויות אלו מגדירות איך יתנהל ה-Session, הנה כמה אופציות לדוגמא:

שם	קוד	משמעות
BINARY	0	שימוש בתעבורה בינארית.
ECHO	1	לאפשר שימוש ב-**Echo**.
NAWS	31	קביעת גודל חלונית
STATUS	5	בקש או שלח סטטוס של אחת האופציות.

- לרשימה המלאה של האופציות, ניתן לצפות במפרט של IANA.

בכדי שהשרת ולקוח ידעו שהמידע המתקבל הוא חלק מהשיח הזה, תמיד ישלח תחילה הבייט **IAC**, ולאחר מכן הפקודה. במידה וצריך, תשלח גם האופציה.

לאחר שנשלח ה-**IAC** האחרון מהשרת, החיבור **Telnet** נוצר בהצלחה!

הטמעת Telnet

חלק זה מתמקד בהטמעה של לקוח Telnet ב-CPP, אך אני בטוח שאם יש לכם קצת רקע בתכנות תוכלו "לתרגם" מדריך זה לכל שפה.

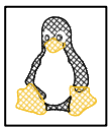
- חשוב לי לציין שלטובת המאמר בחרתי לפשט את הקוד שנדרש כדי ליצור לקוח Telnet, ולמרות שהכל יעבוד כמו שצריך, זה לא יהיה לקוח Telnet מקצועי. הקוד המוצג כאן הוא רק כדי להעביר את השלבים הבסיסים הנדרשים כדי ליצור חיבור, לא להחזיק אותו כשורה.

ראשית אפתח Class חדש, למשל עם השם Session:

```
#ifndef RTELNET_H
#define RTELNET_H

inline constexpr int RTELNET_SUCCESS = 0;
inline constexpr int RTELNET_PORT    = 23;

class session {
 public:
    int _port = RTELNET_PORT;
    const char* _address;
    std::string _username;
    std::string _password;

   session(
      const char* address,
      const std::string username,
      const std::string password,
      int port = RTELNET_PORT
    ) :
      _address(address),
      _username(username),
      _password(password),
      _port(port) {}
 private:
    bool _connected = false;
    bool _negotiated = false;
    bool _logged_in = false;
    int _fd;
};

#endif // RTELNET_H
```

כמו שאתם יודעים (או לא), CPP לא תומך ברשתות תחילה (לא כלול ב-Standard library). זה אומר שחייב להשתמש באיזושהי ספריה חיצונית כדי ליצור את ה-TCP session, לטובת פעולה זו בחרתי להשתמש בספריות הבאות: ספציפי ל-(POSIX/Linux)

- sys/socket.h
- netinet/in.h
- arpa/inet.h



אני אכתוב Class פשוט תחת Session כדי להתנהל עם החיבור TCP, אך לפני זה, נקבע קודי יציאה בעת התקלות בשגיאה:

```
enum Errors {
  ADDRESS_NOT_VALID    = 210,
  CANNOT_ALOCATE_FD    = 211,
  CONNECTION_CLOSED_R = 212,
  NOT_CONNECTED        = 213,
  FAILED_SEND          = 214,
  PARTIAL_SEND         = 215
};
```

כדי להימנע מלהחזיר Exception, גם מבחינת נוחיות כספריה וגם מבחינת מהירות, נחזיר integer. לפי ערך זה המשתמש יכול להתנהל בעת שגיאה:

- **0** או RTELNET_SUCCESS: אומר לנו שהפעולה הצליחה.
- 200 ומטה: מחזיר את שגיאת errno.
- 200 ומעלה: שגיאות בהתאמה אישית.

ופונקציה **inline** המחזירה את השגיאה המלאה:

```
inline std::string_view readError(int rtntErrno) {
  if (rtntErrno < 200) { return strerror(errno); } // return errno
  switch (rtntErrno) {
    case RTELNET_SUCCESS: return              "No error.";
    case Errors::ADDRESS_NOT_VALID: return    "Address is not valid.";
    case Errors::CANNOT_ALOCATE_FD: return    "Cannot allocate a file
descriptor.";
    case Errors::CONNECTION_CLOSED_R: return "Connection closed by remote.";
    case Errors::NOT_CONNECTED: return        "Connection failed.";
    case Errors::FAILED_SEND: return          "Could not send message.";
    case Errors::PARTIAL_SEND: return         "Message was sent partially.";
    default: return                           "Unknown error.";
  }
}
```

- errno הוא משתנה גלובלי במערכות לינוקס ו-POSIX, המשמש לדווח על קוד שגיאה כשהפונקציה האחרונה של הספרייה הסטנדרטית נכשלה.

עכשיו ניתן לכתוב את ה-Class לניהול החיבור TCP תחת Session:

```
class tcp {
public:
  tcp(session* owner) : _owner(owner) {}
private:
  session* _owner;
};
```

פונקצית עזר להגדרת האובייקט sockaddr_in:

```
inline unsigned int setSocketAddr(sockaddr_in& server_address) const {
  server_address.sin_family = AF_INET;
  server_address.sin_port = htons(_owner->_port);

  if (inet_pton(AF_INET, _owner->_address, &server_address.sin_addr) <= 0) {
    return Errors::ADDRESS_NOT_VALID;
  }

  return RTELNET_SUCCESS;
}
```

חיבור על גבי sockaddr_in:

```
inline unsigned int Connect(sockaddr_in& address) {
  int sockfd = socket(AF_INET, SOCK_STREAM, 0);
  if (sockfd < 0) return Errors::CANNOT_ALOCATE_FD;

  errno = 0;
  if (connect(sockfd, reinterpret_cast<sockaddr*>(&address), sizeof(address)) <
0) { return errno; }

  _owner->_connected = true;
  return sockfd;
}
```

סגירה:

```
void Close() {
  close(_owner->_fd);
  _owner->_connected = false;
}
```

שליחה: (בינארי)

```
inline unsigned int SendBin(const std::vector<unsigned char>& message, int
sendFlag = 0) const {
  if (!_owner->_connected) return Errors::NOT_CONNECTED;

  errno = 0;
  ssize_t bytesSent = send(_owner->_fd, message.data(), message.size(),
sendFlag);

  if (bytesSent == 0) return Errors::FAILED_SEND;
  if (static_cast<size_t>(bytesSent) != message.size()) return
Errors::PARTIAL_SEND;
  if (bytesSent < 0) return errno;

  return RTELNET_SUCCESS;
}
```

פונקציה זו נועדה לשלוח הודעות לשרת Telnet כחלק מ-Negotiation או Re-negotiation.

שליחה: (מחרוזות)

```
inline unsigned int Send(const std::string& message, int sendFlag = 0) const {
  if (!_owner->_connected) return Errors::NOT_CONNECTED;

  std::vector<unsigned char> buffer;
  buffer.reserve(message.size());

  // Escape 255/0xFF
  for (unsigned char c : message) {
    buffer.push_back(c);
    if (!(_owner->_binarySendEnabled && _owner->_binaryReceiveEnabled) && c ==
255) buffer.push_back(255);
  }

  errno = 0;
  ssize_t bytesSent = send(_owner->_fd, buffer.data(), buffer.size(),
sendFlag);

  if (bytesSent == 0) return Errors::FAILED_SEND;
  if (bytesSent < 0) return errno;
  if (static_cast<size_t>(bytesSent) != buffer.size()) return
Errors::PARTIAL_SEND;

  return RTELNET_SUCCESS;
}
```

- שימו לב, שכאשר אנו רוצים לשלוח מידע רגיל, יש להפוך כל בייט של 255 אל 255 255, שכן 255 שווה ערך אל **IAC**.

קריאה:

```
inline unsigned int Read(std::vector<unsigned char>& buffer, int readSize =
RTELNET_BUFFER_SIZE, int recvFlag = 0) const {
  if (!_owner->_connected) return Errors::NOT_CONNECTED;

  buffer.resize(readSize);

  fd_set readfds;
  FD_ZERO(&readfds);
  FD_SET(_owner->_fd, &readfds);

  timeval timeout{};
  timeout.tv_sec = 1;
  timeout.tv_usec = 0;

  int ready = select(_owner->_fd + 1, &readfds, nullptr, nullptr, &timeout);
  if (ready < 0) return errno;
  if (ready == 0) {
    buffer.clear();
    return RTELNET_SUCCESS;
  }

  errno = 0;
  ssize_t bytesRead = recv(_owner->_fd, reinterpret_cast<char*>(buffer.data()),
readSize, recvFlag);

  if (bytesRead < 0) return errno;
  if (bytesRead == 0) return Errors::CONNECTION_CLOSED_R;

  buffer.resize(bytesRead);

  return RTELNET_SUCCESS;
}
```

לבסוף נעדכן את Session: (תוסיפו גם את _tcp אל ה-Constructor)

```
  ~session() {
    _tcp.Close();
  }

tcp _tcp;
private:
  friend class tcp;
```

עכשיו שיש לנו את כל הכלים הנדרשים לתקשורת תקינה על גבי פרוטוקול Telnet, אנו יכולים להתחיל לכתוב את הפונקציות שיחזיקו את החיבור.

נעדכן את ה-enum של Errors (ואת הפונקציה מחזירה שגיאות):

```
enum Errors {
  ADDRESS_NOT_VALID    = 210,
  CANNOT_ALOCATE_FD    = 211,
  CONNECTION_CLOSED_R = 212,
  NOT_CONNECTED        = 213,
  FAILED_SEND          = 214,
  PARTIAL_SEND         = 215,
  NOT_A_NEGOTIATION    = 216,
  NOT_NEGOTIATED       = 217,
  USERNAME_NOT_SET     = 218,
  PASSWORD_NOT_SET     = 219,
  NOT_LOGGED           = 220,
  FAILED_LOGIN         = 221
};
```

Negotiate

פונקציה זו אחראית לנהל את שלב ה-Negotiation של Telnet.

לפי שנרשום את הפונקציה, נגדיר enums שיחזיקו את הבייטים אותם נשלח בעת Negotiation:

```
enum TelnetCommands : unsigned char {
  IAC  = 255, // Interpret As Command
  DO   = 253, // Please use this option
  DONT = 254, // Please don't use this option
  WILL = 251, // I will use this option
  WONT = 252  // I won't use this option
};

enum TelnetOptions : unsigned char {
  BINARY        = 0, // Binary transmission (8-bit clean communication)
  ECHO          = 1, // Echo input back to sender (commonly server-side)
  SGA           = 3, // Suppress Go Ahead (stream mode instead of line mode)
  STATUS        = 5, // Query or send current option status
  TIMING_MARK   = 6, // Timing mark for synchronization
  TERMINAL_TYPE = 24 // Exchange terminal type (e.g., "ANSI", "VT100")
  // Ideally add all.
}
```

תחילה, נבדוק אם נוצר חיבור TCP:

```
unsigned int Negotiate() {
  if (!_connected) return Errors::NOT_CONNECTED;
  return RTELNET_SUCCESS;
}
```

במידה וכן, ניצור לולאה בה נקשיב לבקשות מהשרת (בצפייה אל IAC):

```
std::vector<unsigned char> buffer;

while (true) {
  // Peek in the buffer
  buffer.clear();
  unsigned int bufferPeek = _tcp.Read(buffer, 3, MSG_PEEK);
  if (bufferPeek != RTELNET_SUCCESS) { return bufferPeek; }
}
```

אחרי שהצצנו ב- Buffer(בהודעה מהשרת), ניתן לבדוק אם הוא תואם את הצפיות שלנו:

```
ssize_t n = static_cast<ssize_t>(buffer.size());
if (n < 3) { // There are less than 3 bytes (Not IAC COMMAND OPTION)
  _negotiated = false;
  return Errors::NOT_NEGOTIATED;
}

// If not a negotiation packet, exit
if (buffer[0] != TelnetCommands::IAC) {
  if (!_negotiated) {
    return Errors::NOT_A_NEGOTIATION;
  } else {
    break;
  }
}
```

במידה והוא כן תואם את הציפיות שלנו, אנחנו יכולים לקרוא אותו במלואו:

```
buffer.clear();
unsigned int bufferResult = _tcp.Read(buffer, 3);
if (bufferPeek != RTELNET_SUCCESS) return bufferResult;

unsigned char command = buffer[1];
unsigned char option  = buffer[2];
```

ולהחזיר תשובה לשרת:

```
std::vector<unsigned char> response = {
    static_cast<unsigned char>(TelnetCommands::IAC),
    static_cast<unsigned char>(0),
    static_cast<unsigned char>(option)
  };
```

נגדיר את התשובה שלנו: (כרגע אנחנו לא תומכים באף אופציה, התשובה תמיד תהיה שלילית)

```
switch (command) {
  case TelnetCommands::DO:
    switch (option) {
      case TelnetOptions::BINARY: response[1] = TelnetCommands::WONT; break;
      case TelnetOptions::ECHO: response[1] = TelnetCommands::WONT; break;
      case TelnetOptions::SGA: response[1] = TelnetCommands::WONT; break;
      case TelnetOptions::STATUS: response[1] = TelnetCommands::WONT; break;
      case TelnetOptions::TIMING_MARK: response[1] = TelnetCommands::WONT;
break;
      case TelnetOptions::TERMINAL_TYPE: response[1] = TelnetCommands::WONT;
break;
      default: response[1] = TelnetCommands::WONT; break;
    }
    break;

  case TelnetCommands::WILL:
    switch (option) {
      case TelnetOptions::BINARY: response[1] = TelnetCommands::WONT; break;
      case TelnetOptions::ECHO: response[1] = TelnetCommands::WONT; break;
      case TelnetOptions::SGA: response[1] = TelnetCommands::WONT; break;
      case TelnetOptions::STATUS: response[1] = TelnetCommands::WONT; break;
      case TelnetOptions::TIMING_MARK: response[1] = TelnetCommands::WONT;
break;
      case TelnetOptions::TERMINAL_TYPE: response[1] = TelnetCommands::WONT;
break;
      default: response[1] = TelnetCommands::WONT; break;
    }
    break;
  case TelnetCommands::WONT:
  case TelnetCommands::DONT:
    // Ideally, add support for these later.
    break;
}
```

ולסיום נשלח את התשובה חזרה לשרת:

```
  _tcp.SendBin(response);
  _negotiated = true;
```

בעת יציאה מהלולאה (break), נחזיר:

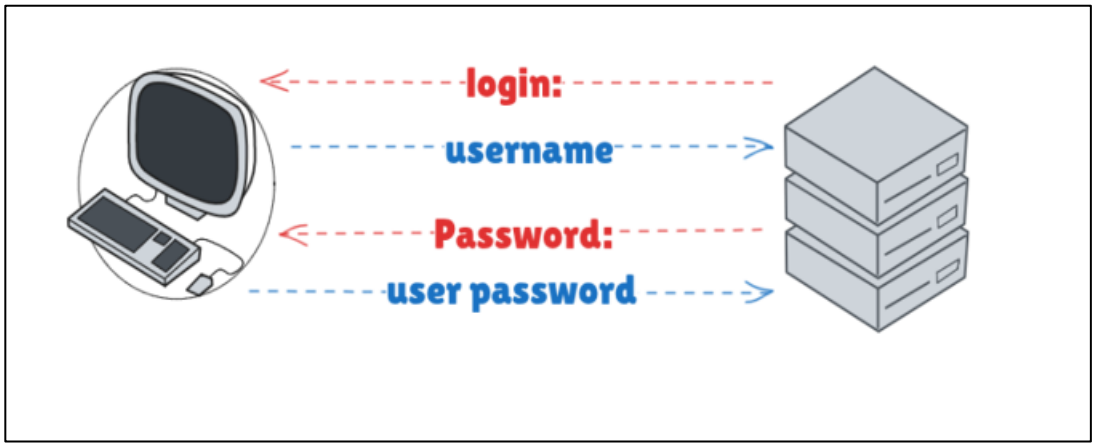
```
return RTELNET_SUCCESS;
```

כדי לסמל **Negotiation** מוצלח.

Login

עכשיו שיש לנו את היכולת ליצור חיבור TCP עם השרת, ולעבור את שלב ה-**Negotiation**, אנו יכולים לנסות להתחבר אל משתמש בשרת.

הליך ההתחברות נראה כך:



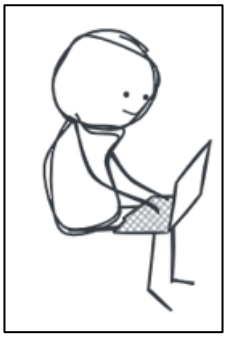
כלומר, השרת שולח מחרוזת ובה המילה "login:", ותפקידנו פשוט להחזיר לשרת את השם משתמש, ואותו הדבר עם הסיסמא.

לטובת פעולה זו, ניצור פונקציית עזר לתפוס ולצפות למחרוזת מסוימת (לדוגמא Login):

```
unsigned int expectOutput(const std::string& expect, std::vector<unsigned
char>& buffer) {
    if (!_connected) return Error::NOT_CONNECTED;
    if (!_negotiated) return Error::NOT_NEGOTIATED;

    for (int i = 0; i < 300; ++i) {
      unsigned int readStatus = _tcp.Read(buffer);
      if (readStatus != RTELNET_SUCCESS) return readStatus;

      std::string cleanedBuffer(reinterpret_cast<const char*>(buffer.data()),
buffer.size());

      cleanedBuffer.erase(std::remove(cleanedBuffer.begin(),
cleanedBuffer.end(), '\r'), cleanedBuffer.end());
      cleanedBuffer.erase(std::remove(cleanedBuffer.begin(),
cleanedBuffer.end(), '\n'), cleanedBuffer.end());

      if (cleanedBuffer.find(expect) != std::string::npos) {
          return RTELNET_SUCCESS;
      }

      std::this_thread::sleep_for(std::chrono::milliseconds(200));
    }

    return Error::CANT_FIND_EXPECTED;
}
```

ונשתמש בה בפונקצית Login:

```
unsigned int Login() {
  if (!_connected) return Error::NOT_CONNECTED;
  if (!_negotiated) return Error::NOT_NEGOTIATED;
  if (_username.empty()) return Error::USERNAME_NOT_SET;
  if (_password.empty()) return Error::PASSWORD_NOT_SET;

  std::vector<unsigned char> buffer;

  // Enter login
  buffer.clear();
  unsigned int loginStatus = expectOutput("login:", buffer);
  if (loginStatus != RTELNET_SUCCESS) return loginStatus;
  unsigned int loginResponse = _tcp.Send(_username + "\n");
  if (loginResponse != RTELNET_SUCCESS) return loginResponse;

  // Enter password
  buffer.clear();
  unsigned int passwordStatus = expectOutput("Password:", buffer);
  if (passwordStatus != RTELNET_SUCCESS) return passwordStatus;
  unsigned int passwordResponse = _tcp.Send(_password + "\n");
  if (passwordResponse != RTELNET_SUCCESS) return passwordResponse;

  _logged_in = true;

  return RTELNET_SUCCESS;
}
```



Connect

מאחר ויש לנו עכשיו איך לעבור את שלב ה-Negotiation ויש לנו איך להתחבר למשתמש, ניצור דרך קלה לקרוא אל פונקציות אלו ממקום אחד:

```
unsigned int Connect() {
  sockaddr_in address;
  unsigned int addressResult = _tcp.setSocketAddr(address);
  if (addressResult != 0 ) return addressResult;

  int fd = _tcp.Connect(address);
  if (fd < 0) return fd;
  _fd = fd;

  int negotiateStatus = Negotiate();
  if (negotiateStatus != RTELNET_SUCCESS) return negotiateStatus;

  return RTELNET_SUCCESS;
}
```

כך שחסכנו למשתמש את שלוש קריאות אל פונקציות שונות.

ביצוע פקודות:

אחרי ש:

- יצרנו **TCP Session**
- עברנו את שלב ה-**Negotiation**
- התחברנו אל משתמש.

אנחנו סוף סוף יכולים לבצע פקודות.

- זכרו שכאשר שולחים פקודה/הודעה יש להכפיל כל בייט של 255, כך שאם יש 255 הוא יראה כך: 255 255 (על פי ה-RFC).

אחת הבעיות הקשות ביותר כאשר מנסים לשלוח פקודות על גבי פרוטוקול כמו Telnet, שנועד לשימוש אנושי ולא על ידי תוכנות, הוא שלרוב אתה לא באמת יודע מתי הסתיימה ביצועה של פקודה.

אני מצאתי שתי אפשרויות לפתור את בעיה זו:

1. לחכות עד שחוזר ה-**prompt** שוב. (לדוגמא:[~root@server]#)
2. לחכות כמות זמן ידועה, בה לא נשלח עוד מידע.

לשתי הפתרונות יש יתרונות וחסרונות:

לחכות ל-**prompt**:

יתרונות	חסרונות
אמין אם הפרומפט עקבי וייחודי	נכשל אם הפרומפט משתנה או מוסתר
מסמן בבירור את סיום הפקודה	נשבר אם הפרומפט מופיע בפלט של הפקודה (למשל ב־cat, echo)
מונע עיכובים מיותרים כאשר הפרומפט מופיע מיד	דורש לוגיקת ניתוח לזיהוי תבניות של פרומפטים
	לא מתאים להתקנים ללא פרומפט יציב או גלוי

המתנה לפי חוסר פעילות (**timeout**):

יתרונות	חסרונות
פועל גם בלי להסתמך על פרומפט	עלול לקטוע פלט של פקודות איטיות או ארוכות
פשוט למימוש חיבורי Telnet או shell כלליים	מוסיף עיכוב מיותר בפקודות מהירות
מונע בעיות ניתוח פרומפט או עיצוב	דורש כוונון מדויק של הזמן כדי לאזן בין מהירות לשלמות

לפי דעתי הסתמכות על המתנה לחוסר פעילות היא אופציה טובה יותר, לכן עדכנתי את **Session** להחזיק את הערכים הבאים:

```
inline constexpr int RTELNET_IDLE_TIMEOUT        = 1000;
inline constexpr int RTELNET_TOTAL_TIMEOUT       = 10000;

// In the Session class:
int _idle = RTELNET_IDLE_TIMEOUT;
int _timeout = RTELNET_TOTAL_TIMEOUT;
```

ויצרתי את הפונקציה הבאה:

```
unsigned int Execute(const std::string& command, std::string& buffer) {
  if (!_connected) return Error::NOT_CONNECTED;
  if (!_negotiated) return Error::NOT_NEGOTIATED;
  if (!_logged_in) return Error::NOT_LOGGED;

  unsigned int sendStatus = _tcp.Send(command + "\n");
  if (sendStatus != RTELNET_SUCCESS) return sendStatus;

  std::vector<unsigned char> output;
  buffer.clear();
```

```
  auto startTime = std::chrono::steady_clock::now();
  auto lastRead = startTime;

  while (true) {
    unsigned int readStatus = _tcp.Read(output);
    if (readStatus != RTELNET_SUCCESS) return readStatus;

    if (!output.empty()) {
      buffer.append(reinterpret_cast<const char*>(output.data()),
output.size());
      lastRead = std::chrono::steady_clock::now();
    }

    auto now = std::chrono::steady_clock::now();
    auto idle = std::chrono::duration_cast<std::chrono::milliseconds>(now -
lastRead).count();
    auto total = std::chrono::duration_cast<std::chrono::milliseconds>(now -
startTime).count();

    if (idle > _idle || total > _timeout) break;

    std::this_thread::sleep_for(std::chrono::milliseconds(50));
  }

  return RTELNET_SUCCESS;
}
```

היא לוקחת כפרמטר את הפקודה ואזכור אל buffer לבחירת המשתמש שאליו נכנס ה-output, ונועדה להמתין למשך זמן חוסר פעילות. כלומר, היא תפסיק לקרוא אם לא התקבלו נתונים חדשים במהלך פרק זמן זה, או אם זמן ההמתנה הכולל חרג מהמגבלה שנקבעה.

שימוש בספריה

עכשיו שיש לנו עם מה לעבוד, ניתן להשתמש בספריה שכתבנו עם תוכנה פשוטה מאוד המחזירה את התוכן של תיקיית /demo בשרת:

```
#include "rtelnet.hpp"
#include <iostream>

int main(int argc, char *argv[]) {
  if (argc < 2) {
    std::cerr << "[ERROR]: Usage: " << argv[0] << " <ADDRESS> <PORT>\n";
    return 1;
  }

  session Session(argv[1], "example", "1234", std::atoi(argv[2]));

  unsigned int connectSuccess = Session.Connect();
  if (connectSuccess != RTELNET_SUCCESS) {
    std::cerr << "[CONNECTION_ERROR]: " << readError(connectSuccess) << "\n";
```

```
    return 1;
  }

  std::string buffer;

  // Flush the banner
  unsigned int fbSuccess = Session.Execute("\n", buffer);
  if (fbSuccess != RTELNET_SUCCESS) {
    std::cerr << "[EXEC_ERROR]: " << rtnt::readError(fbSuccess) << "\n";
    return 1;
  }

  // Execute command
  unsigned int execSuccess = Session.Execute("ls -al /demo", buffer);
  if (execSuccess != RTELNET_SUCCESS) {
    std::cerr << "[EXEC_ERROR]: " << rtnt::readError(execSuccess) << "\n";
    return 1;
  }

  std::cout << buffer << "\n";
  return 0;
}
```

לאחר קומפילציה של הקוד, ניתן לראות שקיבלנו חזרה את תוכן התיקיה \demo בהצלחה:

```
noam ◈ noam ⊛ build ⊛ ❯❯ make
[ 50%] Building CXX object CMakeFiles/relic-telnet.dir/src/rtelnet.cpp.o
[100%] Linking CXX executable /home/noam/Work_spaces/Personal/chronicle/relic-
telnet/bin/relic-telnet-tester
[100%] Built target relic-telnet
noam ◈ noam ⊛ build ⊛ ❯❯ ../bin/relic-telnet-tester 172.16.1.2 23
total 8
drwxr-xr-x   2 root root 4096 Jun 30 08:28 .
-rw-r--r--   1 root root    0 Jun 30 08:28 Hello
-rw-r--r--   1 root root    0 Jun 30 08:27 Digital
-rw-r--r--   1 root root    0 Jun 30 08:27 Whisper
dr-xr-xr-x. 20 root root 4096 Jun 30 08:27 ..
[example@server ~]$
```

סיכום

בניית לקוח **Telnet** מהווה תרגול יעיל ומעמיק להבנת עקרונות יסוד בתכנות רשת, עבודה עם פרוטוקולים, ותכנון ממשק תקשורת יציב.

למרות הפשטות היחסית של פרוטוקול **Telnet**, יישומו בפועל דורש התמודדות עם תזמון, ניתוח תגובות מהשרת, זיהוי תבניות, ניהול מצב חיבור ופרשנות לפקודות ברמת הבייט.

המאמר מציג שלב אחר שלב כיצד ניתן לממש לקוח **Telnet** מודולרי ב-**CPP**, תוך שימוש ביכולות בסיסיות של **POSIX** וספריות מערכת. אך הוא לא מייצג לקוח **Telnet** מוכן לסביבת פרודקשן, ממש לא, הוא מראה מה הבסיס שאיתו צריך לעבוד בעת פיתוח לקוח **Telnet**.

מדריך זה נועד לשמש בסיס למי שמעוניין להבין את פרוטוקול **Telnet** לעומק, לפתח כלים מותאמים אישית, או פשוט לשפר את הבנתו בתקשורת מבוססת **TCP**, ויכול להוות שער כניסה לפיתוח מול פרוטוקולים נוספים ולפרויקטים ברמת מערכת.

- אם אתם בונים משהו ורוצים לכם לשתף, תמיד שמח לראות ;)

על המחבר

סתם אוהב לחקור על דברים 🙃

- LinkedIn
- GitHub

מקורות מידע

- IANA Telnet specification
- Telnet RFC