Katedra Informatyki

Wydział Informatyki i Telekomunikacji
Politechnika Krakowska

Programowanie Usług Sieciowych

mgr inż. Michał Niedźwiecki

SSH

Laboratorium: 07,

system operacyjny: Linux

Kraków, 2022

Spis treści

1. Wiadomości wstępne

Pierwsza część niniejszej instrukcji zawiera podstawowe wiadomości teoretyczne dotyczące protokołu SSH oraz biblioteki libssh2. Poznanie tych wiadomości umożliwi prawidłowe zrealizowanie praktycznej części laboratorium.

1.1. Tematyka laboratorium

Tematyką laboratorium jest programowanie aplikacji wykorzystujących bibliotekę libssh2. Biblioteka libssh2 implementuje protokół SSH2 i jest dostępna na licencji

BSD License. API biblioteki umożliwia:

- uwierzytelnianie serwera za pomocą kluczy RSA lub DSA,
- uwierzytelnianie użytkowników za pomocą metod: none, password, keyboardinteractive, hostbased oraz publickey,
- szyfrowanie za pomocą algorytmów AES, 3DES, Blowfisch, CAST i ARC4,
- tworzenie kanałów typu: shell, exec, subsystem,
- forwarding portów TCP oraz forwarding X11,
- korzystanie z podsystemu SFTP oraz podsystemu klucza publicznego.

Biblioteka libssh2 została zaimplementowana w języku C, ale wiele innych języków programowania – w tym Perl, Python, PHP – udostępnia własne interfejsy (tzw. wrappery), które umożliwiają korzystanie z jej funkcjonalności. Libssh2 wymaga obecności biblioteki OpenSSL lub libgcrypt. Ponadto, biblioteka może zostać opcjonalnie skompilowana z obsługą kompresji zlib.

1.2. Zagadnienia do przygotowania

Przed przystąpieniem do realizacji laboratorium należy zapoznać się z następującymi zagadnieniami dotyczącymi protokołu SSH2: [1 – 10]

- architektura protokołu,
- protokoły Transport Layer Protocol, User Authentication Protocol, Connection Protocol,
- metody wymiany klucza, algorytm Diffiego-Hellmana, Perfect Forward Secrecy
- metody uwierzytelniania użytkowników,
- zasada forwardingu portów TCP (forwarding zdalny i lokalny),
- podsystemy sftp i publickey,
- API biblioteki libssh2 (sesje, uwierzytelnianie, kanały, podsystem SFTP).

Ponadto, wymagana jest: [11 - 14]

- umiejętność konfiguracji serwera OpenSSH,
- znajomość programu ssh-keygen,
- umiejętność korzystania z gniazd TCP i konwersji adresów IP (funkcje inet_pton(), inet_ntop()).

Literatura:

[1] IETF (http://www.ietf.org/), RFC 4251, „The Secure Shell (SSH) Protocol Architecture”

[2] IETF, RFC 4253, „The Secure Shell (SSH) Transport Layer Protocol”

[3] IETF, RFC 4252, „The Secure Shell (SSH) Authentication Protocol”

[4] IETF, RFC 4254, „The Secure Shell (SSH) Connection Protocol”

[5] IETF, RFC 4256, „Generic Message Exchange Authentication for the Secure Shell Protocol (SSH)”

[6] IETF, RFC 4419, „Diffie-Hellman Group Exchange for the Secure Shell (SSH) Transport Layer Protocol”

[7] IETF, RFC 4819, „Secure Shell Public Key Subsystem”

[8] IETF, Internet-Draft („work in progress”), „SSH File Transfer Protocol draft-ietfsecsh-filexfer-13.txt”

[9] Libssh2 (http://libssh2.haxx.se/), „Documentation”

[10] WinSCP (http://winscp.net/), „Supported Transport Protocols”

[11] MAN (5) „sshd_config”

[12] MAN (1) „ssh”, „ssh-keygen”

[13] Michael Stahnke, „Pro OpenSSH”, Apress

[14] W.R. Stevens, „Programowanie Usług Sieciowych”, „API: gniazda i XTI”

1.3. Opis laboratorium

SSH jest protokołem, którego celem jest zapewnienie bezpiecznej komunikacji w sieci Internet. SSH bazuje na architekturze klient-serwer i składa się z trzech zasadniczych komponentów: protokołu Transport Layer Protocol, protokołu User Authentication Protocol oraz protokołu Connection Protocol.

1.3.1. Transport Layer Protocol

Protokół Transport Layer Protocol (TLP) jest odpowiedzialny za uwierzytelnianie serwera oraz zapewnienie poufności i integralności transmitowanych danych. TLP operuje bezpośrednio nad protokołem warstwy transportowej modelu referencyjnego ISO/OSI. Protokół warstwy transportowej powinien być niezawodny i chronić przed błędami podczas transmisji, co wyklucza UDP.

TLP pozwala na negocjację parametrów bezpiecznego połączenia oraz wymianę tajnego klucza.

Parametry połączenia obejmują:

- algorytm wymiany klucza (wspólny dla obu kierunków połączenia):
    - Diffie-Hellman Key Exchange[2] ze stałą grupą (ang. fixed group),
    - Diffie-Hellman Group and Key Exchange[6],
- algorytm klucza publicznego serwera (wspólny dla obu kierunków połączenia):
    - RSA,

    - DSS (algorytm DSA),
    - certyfikaty OpenPGP z kluczem RSA lub DSS,
- algorytm dla szyfrowania symetrycznego, np. AES,
- algorytm MAC, np. HMAC-SHA1,
- algorytm kompresji danych, np. zlib,
- preferowany język.

Algorytmy symetryczne, MAC, algorytmy kompresji danych oraz preferowany język są negocjowane oddzielnie dla każdego kierunku połączenia, tzn. istnieją dwa zestawy algorytmów:

- client to server – zestaw stosowany przez klienta SSH podczas wysyłania danych oraz przez serwer, dla danych odbieranych,
- server to client – zestaw stosowany przez serwer SSH podczas wysyłania danych oraz przez klienta, dla danych odbieranych.

Wymiana tajnego klucza (K) opiera się na algorytmie Diffiego-Hellmana. Algorytm ten chroni przed pasywnymi atakami (np. sniffing) i zapewnia PFC (ang. Perfect Forward Secrecy). Na podstawie wynegocjowanego klucza (K) generowane są:

- klucze dla algorytmów symetrycznych,
- klucze wykorzystywane przez algorytmy MAC (ang. Message Authentication Code),
- wektory inicjalizacyjne dla symetrycznych algorytmów szyfrujących.

Wymienione materiały kryptograficzne są generowane oddzielnie dla każdego kierunku połączenia. Warto odnotować, że RFC 4253 rekomenduje zmianę kluczy po przesłaniu 1 GB danych.

Proces uwierzytelniania serwera wykorzystuje kryptografię klucza publicznego. Każdy serwer powinien mieć parę kluczy: klucz publiczny (PUB) i klucz prywatny (PRIV). Co więcej, serwer może mieć kilka par kluczy, dla różnych algorytmów, np. RSA i DSA.

Podczas wymiany tajnego klucza (K), serwer podpisuje wiadomość wysyłaną do klienta za pomocą klucza prywatnego (PRIV). Klient, po odebraniu wiadomości z tajnym kluczem (K), weryfikuje podpis cyfrowy serwera za pomocą klucza publicznego serwera (PUB), który również znajduje się w przesyłanej wiadomości.

Z uwagi na fakt, że w sieci Internet nie istnieje publiczna infrastruktura umożliwiająca zweryfikowanie autentyczności klucza (PUB) serwera, stosowany jest następujący model zaufania:

- Podczas pierwszego połączenia klienta z serwerem, klient akceptuje klucz publiczny serwera. Klient może sprawdzić autentyczność klucza na podstawie cyfrowego odcisku palca (ang. fingerprint). Weryfikacja cyfrowego odcisku powinna odbywać się poprzez zewnętrzny kanał, np. linię telefoniczną. Sprawdzenie autentyczności klucza zewnętrznym kanałem chroni przed atakami typu man-in-the-middle.
- Po akceptacji klucza publicznego serwera, klucz jest zapisywany w lokalnej bazie klienta. Wpis w bazie wiąże nazwę hosta (serwera) z jego kluczem publicznym.
- Podczas kolejnych połączeń z serwerem, klucz publiczny serwera jest porównywany z kluczem przechowywanym w lokalnej bazie klienta. Jeżeli klucze są takie same,

proces uwierzytelniania serwera jest kontynuowany, tzn. przeprowadzana jest weryfikacja cyfrowego podpisu serwera. W przeciwnym wypadku klient SSH powinien powiadomić użytkownika o niezgodności kluczy.

Protokół TLP jest podatny na ataki DoS (ang. Denial of Service):

- dowolny klient może nawiązać połączenie i przeprowadzić uwierzytelnianie serwera,
- serwer podczas procesu uwierzytelniania generuje podpis cyfrowy. Generowanie podpisu jest zadaniem złożonym obliczeniowo.

Z uwagi na podatność na ataki DoS, RFC 4253 zaleca ograniczenie dostępu do serwera SSH dla wybranych adresów IP lub podsieci. Alternatywnym rozwiązaniem jest wdrożenie mechanizmu port-knocking lub SPA (ang. Single Packet Authorization).

W wyniku działania protokołu TLP, ustanowiona zostaje sesja SSH i obie strony biorące udział w komunikacji dysponują unikatowym identyfikatorem sesji.

Poniżej przedstawiony jest przebieg ustanawiania bezpiecznego połączenia przez protokół Transport Layer Security dla algorytmu wymiany klucza diffie-hellman-groupexchange-sha1:

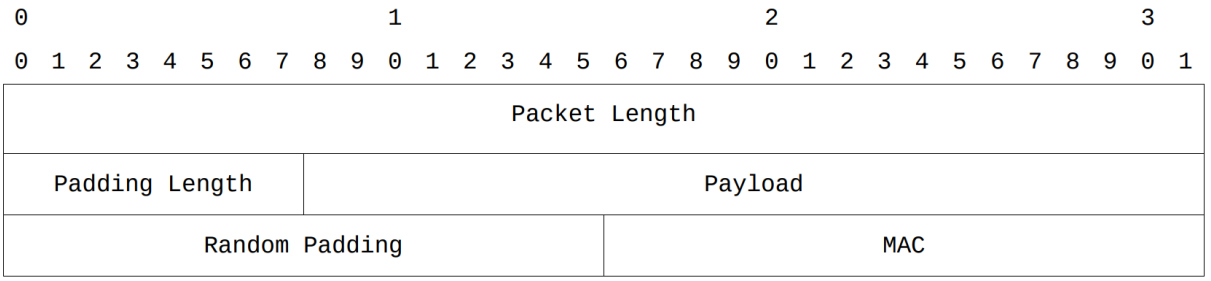
1. Klient ustanawia połączenie TCP z serwerem SSH (three-way handshake).
2. Obie strony muszą przesłać string identyfikujący. Ma on następującą formę:

```
SSH-<wersja SSH>-<nazwa i wersja oprogramowania> <komentarz> CRLF

#Komentarz jest opcjonalny. CR i LF to znaki nowej linii.
#Przykładowo:
SSH-2.0-OpenSSH_3.8.1p1
```

   Wersje protokołu SSH starsze od 2.0 nie zostały udokumentowane i posługują się identyfikatorem postaci: 1.x. Serwer SSH 2.0 kompatybilny ze starszymi wersjami powinien przedstawić wersję 1.99.

3. Wszystkie pakiety wysyłane po przesłaniu stringu identyfikującego wykorzystują Binary Packet Protocol. Pakiet Binary Packet Protocol ma budowę przedstawioną na rys.1.



Rys.1. Budowa pakietu Binary Packet Protocol.

   - Packet Length
     Rozmiar pakietu w bajtach, nie wliczając pola MAC i pola Packet Length.

- Padding Length
  Rozmiar dopełnienia w bajtach; dopełnienie stanowią losowe bajty.
- Payload
  Dane transportowane przez protokół. Jeżeli został wynegocjowany algorytm kompresji danych, to dane są przesyłane w formie skompresowanej (początkowo algorytm kompresji to none). Rozmiar danych wynosi: Packet Length – Padding Length – 1.
- Random Padding
  Losowe bajty dopełnienia. Liczba bajtów musi być dobrana w taki sposób, aby całkowita długość pól: Packet Length, Padding Length, Payload oraz Random Padding była wielokrotnością rozmiaru bloku szyfru symetrycznego lub liczby 8, cokolwiek jest większe. W każdym pakiecie muszą być co najmniej 4 bajty dopełnienia. Maksymalny rozmiar dopełnienia wynosi 255 bajtów. Dopełnienie utrudnia analizę zaszyfrowanej komunikacji na podstawie długości wymienianych komunikatów.
- MAC
  Jeżeli algorytm MAC (ang. Message Authentication Code) został wynegocjowany, pole zawiera kod uwierzytelniający pakietu. Początkowo algorytm MAC to none i pole powinno mieć długość zero.

Minimalny rozmiar pakietu wynosi 16 lub jest równy rozmiarowi bloku szyfru symetrycznego (cokolwiek jest większe), plus rozmiar pola MAC.

Jeżeli algorytmy szyfrujące zostały wynegocjowane, to szyfrowane są wszystkie pola pakietu, oprócz pola MAC.

Kod uwierzytelniający MAC jest obliczany na podstawie klucza, niezaszyfrowanych pól pakietu oraz numeru sekwencyjnego pakietu. Numer sekwencyjny nie jest wysyłany w pakiecie, lecz jest przechowywany lokalnie, oddzielnie dla każdego kierunku połączenia. Pierwszy wysłany pakiet ma numer sekwencyjny równy zero. Numer sekwencyjny jest inkrementowany dla każdego kolejnego pakietu. Wykorzystanie numeru sekwencyjnego do obliczenia MAC chroni przed atakami powtórzeniowymi (ang. replay attacks)[1].

Kod uwierzytelniający MAC jest transmitowany w formie niezaszyfrowanej.

4. Rozpoczyna się proces wymiany klucza. Każda ze stron przesyła pakiet SSH_MSG_KEXINIT, który zawiera m.in.:
   - losowe 16 bajtów, tzw. cookie,
   - listę algorytmów wymiany klucza,
   - listę algorytmów klucza publicznego, które mogą zostać wykorzystane do uwierzytelnienia serwera,
   - listę algorytmów symetrycznych, MAC oraz listę algorytmów kompresji danych dla każdego kierunku połączenia,
   - listę preferowanych języków (może być pusta) dla każdego kierunku połączenia

   Procedurę negocjacji algorytmów opisuje szczegółowo RFC 4253, ale ogólną zasadę można przedstawić jako: wybrany zostanie pierwszy algorytm na liście klienta, który

znajduje się również na liście serwera. Dalsza dyskusja dotyczy przypadku, w którym wynegocjowany został algorytm wymiany klucza o nazwie diffie-hellman-group-exchange-sha1.

5. Serwer przechowuje listę tzw. bezpiecznych liczb pierwszych oraz odpowiadających im generatorów. Liczbę pierwszą p uznaje się za bezpieczną, jeżeli (p1)/2 jest również liczbą pierwszą. Jeżeli serwer nie posiada listy, może generować liczby pierwsze p (ang. prime modulus) i generatory w tle. Liczba pierwsza p oraz generator g stanowią parametry algorytmu Diffiego-Hellmana.
6. Klient wysyła wiadomość SSH_MSG_KEY_DH_GEX_REQUEST, która zawiera:
    - min – minimalny rozmiar liczby pierwszej p w bitach, jaki klient może zaakceptować,
    - n – preferowany przez klienta rozmiar liczby pierwszej p w bitach,
    - max - maksymalny rozmiar liczby pierwszej p w bitach, jaki klient może zaakceptować.
7. Serwer znajduje parametry p oraz g, które najlepiej odpowiadają potrzebom klienta i wysyła je w wiadomości SSH_MSG_KEX_DH_GEX_GROUP.
8. Klient generuje losową liczbę x taką, że: $1 < x < (p – 1)/2$. Następnie klient oblicza $e = g^x \bmod p$ i wysyła liczbę e do serwera za pomocą wiadomości SSH_MSG_KEX_DH_GEX_INIT.
9. Serwer generuje losową liczbę y taką, że $0 < y < (p – 1)/2$ i oblicza:

    $$f = g^y \bmod p.$$

    Po otrzymaniu wartości e, serwer może obliczyć tajny klucz sesyjny:

    $$K = e^y \bmod p = (g^x \bmod p)^y \bmod p = g^{xy} \bmod p = (g^y \bmod p)^x \bmod p = f^x \bmod p$$

    W następnej kolejności, serwer oblicza skrót wiadomości (H) utworzonej z konkatenacji: stringu identyfikacyjnego klienta i serwera (punkt 2), wiadomości SSH_MSG_KEXINIT klienta i serwera (punkt 4), publicznego klucza serwera (KS) oraz parametrów min, n, max, p, g, e, f, K.

    Serwer generuje podpis cyfrowy (S) na podstawie obliczonego skrótu H i wysyła do klienta wiadomość SSH_MSG_KEX_DH_GEX_REPLY, która zawiera:

    - klucz publiczny serwera (KS),
    - wartość $f = g^y \bmod p$,
    - podpis cyfrowy S.

10. Klient weryfikuje klucz publiczny serwera (KS) na podstawie lokalnej bazy kluczy (klient może zaakceptować klucz serwera bez weryfikacji). Następnie klient oblicza tajny klucz sesyjny $K = f^x \bmod p$ i generuje skrót H w sposób identyczny do opisanego w punkcie 9. Dysponując skrótem H i kluczem publicznym serwera, klient może zweryfikować podpis cyfrowy serwera.
11. Rozpoczyna się proces uwierzytelniania użytkownika za pomocą protokołu User Authentication Protocol.

1.3.2. User Authentication Protocol

Protokół User Authentication Protocol (UAP) jest odpowiedzialny za uwierzytelnianie użytkownika. UAP jest protokołem, z którego usług mogą korzystać dowolne algorytmy i mechanizmy uwierzytelniające (nie narzuca konkretnej metody). Wiadomości tego protokołu są transmitowane za pomocą Transport Layer Protocol i UAP zakłada, że transmisja odbywa się z zapewnieniem poufności i integralności danych. W przeciwieństwie do innych popularnych protokołów, np. SSL, proces uwierzytelniania użytkownika nie jest inicjowany przez serwer SSH, ale przez klienta. Serwer SSH nie wymusza zastosowania konkretnej metody i nie żąda przedstawienia danych uwierzytelniających właściwych danej metodzie (hasła, klucza publicznego, itp.). Klient ma wolną rękę – może wybrać dowolną metodę, przesłać dane uwierzytelniające do serwera i oczekiwać na odpowiedź. Jeżeli wybrana przez klienta metoda jest zaimplementowana i skonfigurowana na serwerze, oraz jeżeli przedstawione dane uwierzytelniające są poprawne, serwer akceptuje połączenie. Po zaakceptowaniu połączenia, klient może korzystać z usług protokołu Connection Protocol. W przeciwnym wypadku serwer odrzuca żądanie klienta, po czym klient może próbować uwierzytelniania za pomocą innej metody.

Nazwy metod uwierzytelniających protokołu UAP (podobnie jak nazwy algorytmów wymiany klucza, algorytmów szyfrujących, MAC i algorytmów kompresji danych w protokole TLP) są zdefiniowane w kodowaniu ASCII. Długość nazwy algorytmu nie może przekroczyć 64 znaków. Nazwy algorytmów i metod, które nie posiadają znaku „@” są zarezerwowane przez organizację IANA. Każdy może zdefiniować własne algorytmy i nadać im nazwę w formie: „nazwa_algorytmu@nazwa_domenowa”.

RFC 4252 definiuje następujące metody uwierzytelniania użytkowników: none, hostbased, password, publickey. Dodatkowa metoda - keyboard-interactive – jest zdefiniowana w RFC 4256.

Poniżej przedstawiona jest krótka charakterystyka wymienionych metod:

- none
  Serwer umożliwia ustanowienie połączenia bez uwierzytelniania. W przypadku, gdy serwer nie zezwala na dostęp bez uwierzytelniania, metoda none może zostać wykorzystana do określenia metod wspieranych przez serwer (w odpowiedzi na wiadomość klienta, serwer przesyła listę dostępnych metod).
- hostbased
  Uwierzytelnianie jest przeprowadzane na podstawie:
  - klucza publicznego należącego do hosta, z którego łączy się użytkownik,
  - nazwy hosta, z którego łączy się użytkownik,
  - nazw użytkownika (na stacji klienta i serwera).

  Klient generuje podpis cyfrowy za pomocą klucza prywatnego i przesyła wiadomość z podpisem cyfrowym do serwera. Wiadomość zawiera nazwę użytkownika na stacji serwera (US) oraz nazwę użytkownika na stacji, z której użytkownik nawiązuje połączenie (UK). Użytkownik musi posiadać konto (US) na stacji serwera. Serwer weryfikuje podpis za pomocą klucza publicznego klienta (klucz publiczny klienta musi być w posiadaniu serwera), i na podstawie lokalnej polityki określa, czy połączenie z określonego adresu IP jest dozwolone dla użytkownika (UK).
  Proszę zwrócić uwagę na fakt, że klucz publiczny należy do stacji sieciowej, z której

łączy się użytkownik, a nie do użytkownika. Dowolny użytkownik na stacji sieciowej klienta może wykorzystać klucz publiczny do nawiązania połączenia. Metoda hostbased zakłada, że użytkownik, który nawiązuje połączenie został uwierzytelniony przez stację klienta. Metoda jest stosowana najczęściej w przypadku klastrów obliczeniowych.

- password
  Uwierzytelnianie na podstawie hasła podanego przez użytkownika. Jeżeli hasło użytkownika jest nieaktualne, metoda umożliwia jego zmianę.
- keyboard-interactive
  Klient SSH inicjuje proces uwierzytelniania, ale nie przesyła danych uwierzytelniających. W odpowiedzi na wiadomość klienta, serwer wysyła dowolną liczbę żądań uwierzytelniających (żądania zawierają instrukcje dla użytkownika i opcjonalne dane). Użytkownik odpowiada na każde żądanie serwera stosując się do przesłanych instrukcji. Przykładowo, serwer może zażądać wprowadzenia hasła użytkownika. Jeżeli hasło okaże się prawidłowe, ale nieaktualne (zgodnie z lokalną polityką serwera), to serwer może przesłać kolejne żądanie z prośbą o zmianę hasła. W systemach Linux, metoda keyboard-interactive może zostać wykorzystana z mechanizmem PAM (ang. Pluggable Authentication Modules).
  Inny przykład:
  Klient otrzymuje wraz z żądaniem serwera losowy ciąg cyfr. Użytkownik wprowadza ciąg cyfr do urządzenia przenośnego, które jest odpowiedzialne za wygenerowanie hasła jednorazowego. Otrzymane hasło jest wysyłane do serwera. Metoda keyboard-interactive z łatwością wspiera mechanizmy Challengeresponse oraz One Time Password.
  Dzięki zastosowaniu metody keyboard-interactive, dodanie nowych metod uwierzytelniania nie pociąga za sobą modyfikacji kodu źródłowego klienta SSH. Serwer nie wymaga zmian w kodzie źródłowym, jeżeli wykorzystuje oddzielną warstwę uwierzytelniającą (np. PAM).
- publickey
  Klient wysyła do serwera wiadomość zawierającą m.in.:
  - nazwę użytkownika,
  - klucz publiczny użytkownika,
  - podpis cyfrowy wygenerowany na podstawie wysyłanej wiadomości.

  Po otrzymaniu wiadomości, serwer musi sprawdzić, czy otrzymany klucz publiczny znajduje się w lokalnej bazie kluczy i czy należy do uwierzytelnianego użytkownika. Jeżeli klucz zostanie odnaleziony, serwer weryfikuje podpis cyfrowy klienta. Poprawna weryfikacja oznacza, że użytkownik, który wysłał wiadomość jest w posiadaniu klucza prywatnego odpowiadającego odebranemu kluczowi publicznemu. Fakt posiadania klucza prywatnego pozwala metodzie publickey na uwierzytelnienie użytkownika.

  Serwer SSH po zweryfikowaniu poprawności klucza i podpisu cyfrowego, może wykorzystać dodatkowe mechanizmy uwierzytelniające lub autoryzujące dostęp (nie jest to sprzeczne ze specyfikacją protokołu). Przykładowo, serwer OpenSSH pozwala określić jakie polecenia może wywołać użytkownik, ograniczyć dostęp na podstawie adresu IP klienta, czy zablokować forwarding dla portów TCP. Restrykcje te są związane z kluczem publicznym użytkownika, a nie jego kontem.

RFC 4252 wymaga, aby metoda publickey była obsługiwana przez każdą implementację SSH. Klient SSH może przesłać klucz publiczny utworzony za pomocą dowolnego algorytmu asymetrycznego (algorytm klucza publicznego nie podlega negocjacji). Jeżeli serwer nie obsługuje wybranego algorytmu, powiadomi klienta za pomocą odpowiedniej wiadomości.

Użytkownik może posiadać więcej niż jedną parę kluczy:

- oddzielne klucze dla każdej ze stacji, z której łączy się użytkownik,
- klucze, które użytkownik może powierzyć innym osobom, aby mogły one korzystać z konta użytkownika.

1.3.3. Connection Protocol

Po uwierzytelnieniu, użytkownik ma dostęp do usług protokołu Connection Protocol (CP). CP pozwala na tworzenie kanałów komunikacyjnych w szyfrowanym i uwierzytelnionym tunelu SSH. Dowolna strona (klient lub serwer) może utworzyć kanał komunikacyjny. Kanał jest identyfikowany przez liczby na jego końcach. Należy pamiętać, że liczby odnoszące się do danego kanału mogą być różne na każdym jego końcu. Podczas ustanawiania kanału, obie strony wymieniają się lokalnymi identyfikatorami. Kanały zapewniają kontrolę przepływu – każda strona określa rozmiar danych (tzw. okno) jaki jest w stanie zaakceptować. Odebranie danych powoduje zmniejszenie rozmiaru okna. Rozmiar okna musi być modyfikowany za pomocą specjalnej wiadomości, która pozwala drugiej stronie na przesłanie N bajtów więcej w stosunku do obecnego rozmiaru okna.

Większość wiadomości protokołu CP wiąże się z tworzeniem kanałów komunikacyjnych lub z transmisją danych przez wybrane kanały. Istnieją jednak wiadomości, które wpływają na komunikujące się strony niezależnie od kanałów komunikacyjnych. Są to tzw. globalne zapytania/prośby (ang. global requests). Przykładem takiej wiadomości jest prośba rozpoczęcia zdalnego forwardingu portów TCP (tcpip-forward), wysyłana przez klienta SSH. Po otrzymaniu takiej wiadomości, serwer może rozpocząć nasłuchiwanie na wybranym przez klienta porcie. Dla każdego zaakceptowanego połączenia TCP, serwer tworzy kanał typu forwarded-tcpip, obsługujący dane połączenie TCP.

Lokalny forwarding portów (direct-tcpip) nie wymaga wysłania globalnego zapytania. W tym przypadku dowolna strona może utworzyć gniazdo nasłuchujące, zaakceptować połączenie TCP i utworzyć kanał do jego obsługi. Zdalna strona nie musi zgodzić się na utworzenie kanału direct-tcpip. Ze względów bezpieczeństwa, klient SSH powinien odrzucać globalne prośby o forwarding portów TCP (tcpip-forward) i nie akceptować kanałów typu direct-tcpip.

Każde zdalne wywołanie programu – powłoki (shell), aplikacji lub polecenia systemowego (exec), podsystemu (subsystem) – określa się mianem sesji. Sesja nie jest niczym więcej, jak kanałem typu session. Nazwy: shell, exec oraz subsystem, stanowią typy zapytań/próśb (ang. channel requests) wysyłane w ramach kanału session. Kanał może obsługiwać tylko jedno z wymienionych zapytań. Ponadto, zapytania te powinny być ignorowane przez klientów SSH. Kanały dla forwardingu portów TCP są niezależne od jakiejkolwiek sesji – zamknięcie sesji nie zamyka połączeń TCP.

Proszę zwrócić uwagę na różnicę między sesją ustanawianą przez protokół Transport Layer Protocol, a sesją protokołu Connection Protocol. W przypadku protokołu TLP, sesja dotyczy połączenia SSH i wszystkich kanałów w jego obrębie (w tym sesji CP). Transmisja danych w

sesji może odbywać się za pomocą wiadomości SSH_MSG_CHANNEL_DATA lub SSH_MSG_CHANNEL_EXTENDED_DATA. Pierwsza wiadomość pozwala określić tylko numer kanału odbiorcy. Wiadomość SSH_MSG_CHANNEL_EXTENDED_DATA pozwala dodatkowo zdefiniować typ transmitowanych danych. Do tego celu, nagłówek wiadomości rezerwuje pole o długości 4 bajtów, które można traktować jak zmienną unsigned int. Pole to pozwala na zdefiniowanie aż 232 typów wiadomości. Każdy typ można traktować jako niezależny strumień danych. RFC 4254 rezerwuje wartość 1 dla standardowego wyjścia błędów (SSH_EXTENDED_DATA_STDERR).

Oprócz opisanej funkcjonalności, protokół Connection Protocol pozwala na:

- przekazywanie zmiennych środowiskowych zdalnej powłoce,
- dostarczanie sygnału do zdalnego procesu/usługi,
- określenie statusu zakończenia zdalnego procesu.

1.3.4. Podsystemy

Podsystem jest usługą, protokołem lub programem, który można wywołać za pomocą predefiniowanej nazwy. Przykładem podsystemów są: sftp oraz publickey. SFTP jest protokołem umożliwiającym transmisję plików i przeprowadzanie operacji dyskowych (np. wyświetlanie zawartości katalogu). Błędem jest określanie SFTP, jako „protokołu zapewniającego bezpieczną transmisję plików" – SFTP nie zapewnia bezpieczeństwa, ale korzysta bezpośrednio z usług protokołu SSH. Klient SSH może zażądać skorzystania z usług protokołu SFTP za pomocą nazwy podsystemu: „sftp". Serwer jest odpowiedzialny za obsługę żądania – w tym celu wywołuje zewnętrzną aplikację (serwer SFTP), albo korzysta z wbudowanego serwera SFTP (tzn. usługi wbudowanej w serwer SSH). OpenSSH pozwala na powiązanie nazwy podsystemu z zewnętrzną aplikacją lub wbudowaną usługą za pomocą opcji Subsystem. Poniżej przedstawiona jest postać opcji Subsystem.

```
Subsystem <nazwa podsystemu> <lokalizacja aplikacji/nazwa usługi>
```

Przykłady zastosowania opcji Subsystem dla podsystemu sftp:

```
#Zewnętrzna aplikacja obsługująca protokół SFTP:
Subsystem sftp /usr/local/sbin/sftp-server

#Wbudowany serwer SFTP:
Subsystem sftp internal-sftp

#Możliwe jest zastosowanie tylko jednej z przedstawionych powyżej opcji
```

Podsystem (protokół) publickey umożliwia użytkownikowi zarządzanie własnymi kluczami publicznymi po stronie serwera SSH. Podsystem publickey definiuje możliwe do przeprowadzenia operacje, format wiadomości protokołu SSH dla tych operacji oraz format w jakim muszą być transmitowane klucze publiczne użytkownika. Dzięki temu, użytkownik może zarządzać kluczami w sposób niezależny od implementacji serwera i formatu w jakim serwer przechowuje klucze. Protokół publickey definiuje obecnie 4 operacje:

- dodanie klucza wraz z atrybutami,
- usunięcie klucza,

- pobranie listy kluczy użytkownika,
- pobranie listy atrybutów obsługiwanych przez serwer SSH.

Atrybuty klucza pełnią funkcję autoryzacyjną – określają co wolno użytkownikowi danego klucza, a co jest zabronione. Atrybuty mogą ograniczać dostęp na podstawie adresu IP lub nazwy hosta klienta.

1.3.5. Biblioteka libssh2

API libssh2 udostępnia 7 grup funkcji, które są odpowiedzialne za:

- obsługę sesji protokołu Transport Layer Security (funkcje z prefiksem „libssh2_session_”),
- uwierzytelnianie użytkowników (funkcje z prefiksem „libssh2_userauth_”),
- zarządzanie kanałami komunikacyjnymi (funkcje z prefiksem „libssh2_channel_”),
- podsystem SFTP (funkcje z prefiksem „libssh2_sftp_”),
- podsystem klucza publicznego (funkcje z prefiksem „libssh2_publickey_”),
- zarządzanie kluczami publicznymi użytkownika (funkcje z prefiksem „libssh2_knowhost_”, API dostępne od wersji 1.1.1),
- funkcjonalność niezwiązaną z wymienionymi grupami funkcji (np. libssh2_poll(), libssh2_banner_set()).

Następująca tabela prezentuje jakie pliki nagłówkowe należy załączyć, aby móc korzystać z określonej grupy funkcji.

Grupa funkcji	Plik nagłówkowy
„libssh2_session_” „libssh2_userauth_” „libssh2_channel_” „libssh2_knowhost_” „libssh2_”	<libssh2.h>
„libssh2_sftp_”	<libssh2_sftp.h>
„libssh2_publickey_”	<libssh2_publickey.h>

Laboratorium nie obejmuje zagadnień związanych z podsystemem klucza publicznego oraz zarządzaniem bazą kluczy publicznych użytkownika po stronie serwera.

1.3.5.1. Obsługa sesji protokołu Transport Layer Protocol

Ustanowienie bezpiecznego połączenia za pomocą protokołu Transport Layer Protocol wymaga inicjalizacji i opcjonalnej konfiguracji obiektu sesji, reprezentowanego przed strukturę LIBSSH2_SESSION. W celu utworzenia obiektu LIBSSH2_SESSION, należy wywołać makro libssh2_session_init():

```
LIBSSH2_API LIBSSH2_SESSION *libssh2_session_init(void);
```

Makro libssh2_session_init() jest odpowiedzialne za dynamiczną alokację pamięci dla obiektu sesji. Wiele funkcji libssh2 posiada makra, które wywołują funkcje biblioteki z domyślnymi argumentami. W dalszej części instrukcji makra będą traktowane jak funkcje biblioteki.

Sesję (w rozumieniu protokołu Transport Layer Protocol) można skonfigurować za pomocą funkcji libssh2_session_method_pref(). Funkcja ta pozwala zdefiniować listy preferowanych algorytmów. Listy algorytmów zostaną wykorzystane podczas negocjacji parametrów połączenia.

Poniżej przedstawiony jest prototyp funkcji libssh2_session_method_pref().

```
int libssh2_session_method_pref(LIBSSH2_SESSION *session,
    int method_type,
    const char *prefs);
```

Parametr	Opis
session	wskaźnik na obiekt sesji
method_type	typ konfigurowanej listy algorytmów (jedna ze stałych LIBSSH2_METHOD_* zdefiniowanych w pliku nagłówkowym <libssh2.h>)
prefs	lista preferowanych algorytmów – nazwy algorytmów oddzielone są przecinkami; jeżeli nazwa algorytmu nie jest obsługiwana przez *libssh2*, to jest ona ignorowana

Poniższa tabela przedstawia możliwe wartości parametru method_type.

Wartość	Typ listy
LIBSSH2_METHOD_KEX	lista metod wymiany klucza
LIBSSH2_METHOD_HOSTKEY	lista algorytmów klucza publicznego do uwierzytelniania serwera
LIBSSH2_METHOD_CRYPT_CS	lista szyfrów symetrycznych, *client to server*
LIBSSH2_METHOD_CRYPT_SC	lista szyfrów symetrycznych, *server to client*
LIBSSH2_METHOD_MAC_CS	lista metod MAC, *client to server*
LIBSSH2_METHOD_MAC_SC	lista metod MAC, *server to client*
LIBSSH2_METHOD_COMP_CS	lista algorytmów kompresji danych, *client to server*
LIBSSH2_METHOD_COMP_SC	lista algorytmów kompresji danych, *server to client*
LIBSSH2_METHOD_LANG_CS	lista preferowanych języków, *client to server;* ignorowana przez bibliotekę *libssh2* 1.1
LIBSSH2_METHOD_LANG_SC	lista preferowanych języków, *server to client*; ignorowana przez bibliotekę *libssh2* 1.1

Funkcja libssh2_session_method_pref() zwraca zero w przypadku powodzenia, wartość ujemną w przypadku błędu.

Konfiguracja preferowanych algorytmów jest opcjonalna. Jeżeli funkcja libssh2_session_method_pref() nie zostanie wywołana, to zostaną wykorzystane domyślne ustawienia.

Oprócz listy algorytmów, biblioteka libssh2 umożliwia skonfigurowanie stringu identyfikacyjnego, wysyłanego podczas ustanawiania połączenia (podrozdział 1.3.1).

Konfiguracja stringu identyfikacyjnego jest możliwa za pomocą funkcji libssh2_banner_set().

```
int libssh2_banner_set(LIBSSH2_SESSION *session, const char *banner);
```

Parametr	Opis
session	wskaźnik na obiekt sesji
banner	string identyfikacyjny

Funkcja libssh2_banner_set() zwraca zero w przypadku powodzenia lub wartość ujemną, w przypadku błędu.

Negocjacja parametrów bezpiecznego połączenia SSH rozpoczyna się w momencie wywołania funkcji libssh2_session_startup():

```
int libssh2_session_startup(LIBSSH2_SESSION *session, int socket);
```

Parametr	Opis
session	wskaźnik na obiekt sesji
socket	deskryptor gniazda połączonego dla niezawodnego protokołu warstwy transportowej (TCP)

Jeżeli negocjacja parametrów zakończy się powodzeniem, to funkcja zwróci wartość zero. Powodzenie negocjacji oznacza, że:

- wersje protokołów klienta i serwera SSH są zgodne,
- uzgodnione zostały algorytmy kryptograficzne,
- uzgodniony został tajny klucz sesyjny (za pomocą wybranej metody wymiany klucza),
- serwer przesłał klientowi swój klucz publiczny oraz podpis cyfrowy utworzony za pomocą klucza prywatnego,
- klient odebrał klucz publiczny serwera i zweryfikował za jego pomocą podpis cyfrowy.

Jeżeli klucz publiczny serwera znajduje się w lokalnej bazie kluczy klienta, to otrzymany klucz jest uznawany za zaufany i proces uwierzytelniania serwera kończy się powodzeniem. W przypadku, gdy klucz publiczny serwera nie znajduje się w lokalnej bazie lub klient nie utrzymuje bazy kluczy, to rekomendowana jest weryfikacja autentyczności otrzymanego klucza. Weryfikacji można dokonać za pomocą cyfrowego odcisku palca (ang. fingerprint) obliczonego na podstawie otrzymanego klucza. Biblioteka libssh2 udostępnia funkcję libssh2_hostkey_hash(), która dla otrzymanego klucza publicznego serwera zwraca odcisk palca utworzony za pomocą wybranej funkcji haszującej.

```
const char *libssh2_hostkey_hash(LIBSSH2_SESSION *session,
    int hash_type);
```

Parametr	Opis
session	wskaźnik na obiekt sesji
hash_type	LIBSSH2_HOSTKEY_HASH_MD5 lub LIBSSH2_HOSTKEY_HASH_SHA1

Zwracany jest wskaźnik do tablicy bajtów, a nie string zakończony znakiem '\0'. Długość zwróconego odcisku palca zależy od wybranej funkcji skrótu (16 bajtów dla MD5, 20 bajtów dla SHA-1). Odcisk palca należy zweryfikować za pomocą zewnętrznego kanału, np. linii telefonicznej. Programista nie musi zwalniać pamięci okupowanej przez cyfrowy odcisk (odcisk jest przechowywany w obiekcie sesji i jest usuwany podczas zwalniania pamięci dla obiektu sesji za pomocą libssh2_session_free()).

Nazwy wynegocjowanych algorytmów można uzyskać za pomocą funkcji libssh2_session_methods():

```
const char *libssh2_session_methods(LIBSSH2_SESSION *session,
    int method_type);
```

Parametr Opis

Parametr	Opis
session	wskaźnik na obiekt sesji
method_type	określa typ listy algorytmów; równoważny z parametrem method_type funkcji libssh2_session_method_pref()

Funkcja libssh2_session_methods() zwróci wartość NULL, jeżeli sesja nie została ustanowiona za pomocą libssh2_session_startup().

Jeżeli zachodzi konieczność zdefiniowania trybu operacji I/O dla gniazda TCP wykorzystywanego przez sesję, to można posłużyć się funkcją libssh2_session_set_blocking():

```
void libssh2_session_set_blocking(LIBSSH2_SESSION *session,
    int blocking);
```

Parametr	Opis
session	wskaźnik na obiekt sesji
blocking	wartość zero, jeżeli operacje I/O powinny być blokujące, w przeciwnym wypadku wartość różna od zera; informacje na temat operacji blokujących i nieblokujących można znaleźć w [14]

Zamknięcia sesji można dokonać za pomocą funkcji libssh2_session_disconnect():

```
int libssh2_session_disconnect(LIBSSH2_SESSION *session,
    const char *description);
```

Parametr	Opis
`session`	wskaźnik na obiekt sesji
`description`	przyczyna zamknięcia sesji; tekst jest wysyłany do serwera SSH

W przypadku, gdy sesja została zamknięta i obiekt sesji nie będzie więcej wykorzystywany, można zwolnić pamięć zaalokowaną dynamicznie przez libssh2_session_init(). W tym celu należy wywołać funkcję libssh2_session_free():

```
int libssh2_session_free(LIBSSH2_SESSION *session);
```

Parametr	Opis
`session`	wskaźnik na obiekt sesji zaalokowany przez `libssh2_session_init()`

Zarówno funkcja libssh2_session_disconnect(), jak i libssh2_session_free(), zwracają wartość zero w przypadku powodzenia.

W praktycznej części laboratorium, przydatne mogą okazać się funkcje libssh2_session_last_errno() oraz libssh2_session_last_error():

```
int libssh2_session_last_errno(LIBSSH2_SESSION *session);

int libssh2_session_last_error(LIBSSH2_SESSION *session, char **errmsg,
    int *errmsg_len, int want_buf);
```

Pierwsza z nich zwraca kod ostatniego błędu, związanego z obiektem wskazywanym przez parametr session. Kody błędów zdefiniowane są w pliku nagłówkowym <libssh2.h>. Druga z wymienionych funkcji pozwala pobrać tekstowy opis ostatniego błędu. Parametry funkcji libssh2_session_last_error() zostały omówione w poniższej tabeli.

Parametr	Opis
`session`	wskaźnik na obiekt sesji
`errmsg`	adres wskaźnika na początek stringu; po wywołaniu funkcji, wskaźnik będzie odnosił się do tekstowego opisu błędu; funkcja jest odpowiedzialna za alokację pamięci dla zwracanego stringu (programista dostarcza tylko adres wskaźnika)
`errmsg_len`	po wywołaniu funkcji, zmienna będzie zawierać rozmiar stringu; string jest zakończony znakiem `'\0'` i jego rozmiar można również określić za pomocą funkcji `strlen()`
`want_buf`	jeżeli argument ma wartość różną od zera, to tekstowy opis błędu zwrócony za pomocą parametru `errmsg` musi zostać zwolniony przez programistę; w przeciwnym wypadku pamięcią zarządza biblioteka *libssh2* i programista nie powinien zwalniać pamięci dla stringu

1.3.5.2. Uwierzytelnianie użytkownika

API biblioteki libssh2 implementuje wszystkie metody uwierzytelniania użytkowników omówione w podrozdziale 1.3.2. Do wykonania części praktycznej konieczna jest znajomość następujących funkcji:

- libssh2_userauth_list(),
- libssh2_userauth_password(),
- libssh2_userauth_authenticated(),
- libssh2_userauth_publickey_fromfile().

Funkcja libssh2_userauth_list() wysyła do serwera wiadomość żądającą uwierzytelniania za pomocą metody none. W odpowiedzi na wiadomość, klient powinien otrzymać listę metod uwierzytelniających obsługiwanych przez serwer SSH. Prototyp funkcji przedstawiony jest poniżej.

```
char *libssh2_userauth_list(LIBSSH2_SESSION *session,
    const char *username,
    unsigned int username_len);
```

Parametr	Opis
session	wskaźnik na obiekt sesji
username	nazwa użytkownika
username_len	długość stringu reprezentującego nazwę użytkownika

W przypadku powodzenia, funkcja zwraca listę dostępnych metod uwierzytelniania (nazwy oddzielone przecinkiem). Jeżeli serwer umożliwia dostęp za pomocą metody none lub wywołanie funkcji zakończy się błędem, to zwrócona zostanie wartość NULL.

Błąd można odróżnić od powodzenia za pomocą funkcji libssh2_userauth_authenticated():

```
int libssh2_userauth_authenticated(LIBSSH2_SESSION *session);
```

Funkcja zwraca wartość 1, jeżeli uwierzytelnianie użytkownika zakończyło się powodzeniem, w przeciwnym wypadku - wartość 0.

Proszę zwrócić uwagę na fakt, że proces uwierzytelniania można rozpocząć bez określenia listy metod obsługiwanych przez serwer SSH.

Funkcja libssh2_userauth_password()umożliwia przeprowadzenie uwierzytelniania za pomocą hasła:

```
int libssh2_userauth_password(LIBSSH2_SESSION *session,
    const char *username,
    const char *password);
```

Parametr	Opis
session	wskaźnik na obiekt sesji
username	nazwa użytkownika
username_len	hasło użytkownika

Powodzenie sygnalizowane jest za pomocą wartości zero, błąd za pomocą wartości ujemnej.

Ostatnią z wymienionych funkcji uwierzytelniających jest libssh2_userauth_publickey_fromfile():

```
int libssh2_userauth_publickey_fromfile(LIBSSH2_SESSION *session,
    const char *username,
    const char *publickey,
    const char *privatekey,
    const char *passphrase);
```

Parametr	Opis
session	wskaźnik na obiekt sesji
username	nazwa użytkownika
publickey	ścieżka do pliku z kluczem publicznym użytkownika
privatekey	ścieżka do pliku z kluczem prywatnym użytkownika
passphrase	hasło zabezpieczające dostęp do klucza prywatnego użytkownika

Funkcja libssh2_userauth_publickey_fromfile() przeprowadza uwierzytelnianie użytkownika za pomocą metody publickey. Jeżeli proces uwierzytelniania zakończy się powodzeniem, libssh2_userauth_publickey_fromfile() zwraca wartość zero.

Format klucza prywatnego jest określony przez standard PEM (ang. Privacy Enhanced Mail). Format klucza publicznego jest częściowo zgodny ze standardem stosowanym przez OpenSSH dla pliku authorized_files. Klucz publiczny jest zapisany w jednej linii postaci:

```
<typ klucza> <klucz zakodowany za pomocą Base64> <opcjonalny komentarz>
```

W stosunku do standardu stosowanego dla pliku authorized_files, format klucza publicznego wyklucza zastosowanie wiodącego pola opcji.

Parę kluczy użytkownika można wygenerować za pomocą programu ssh-keygen, który wchodzi w skład projektu OpenSSH.

1.3.5.3. Kanały komunikacyjne

Biblioteka libssh2 udostępnia bogaty interfejs funkcji do tworzenia i zarządzania kanałami komunikacyjnymi.

W celu utworzenia kanału komunikacyjnego można posłużyć się funkcją libssh2_channel_open_ex():

```
LIBSSH2_CHANNEL *libssh2_channel_open_ex(LIBSSH2_SESSION *session,
    const char *channel_type,
    unsigned int channel_type_len,
    unsigned int window_size,
    unsigned int packet_size,
    const char *message,
    unsigned int message_len);
```

Parametr	Opis
session	wskaźnik na obiekt sesji
channel_type	typ kanału; najczęściej jest to jedna z następujących nazw: • "session", • "direct-tcpip", • "tcpip-forward"
channel_type_len	rozmiar nazwy określającej typ kanału w bajtach
window_size	rozmiar okna (maksymalna liczba bajtów jakie zdalna stacja może wysłać przed otrzymaniem wiadomości modyfikującej rozmiar okna)
packet_size	maksymalny rozmiar pakietu z danymi jaki zdalna stacja może wysłać
message	dodatkowe dane, jeżeli są wymagane przez kanał wybranego typu
message_len	rozmiar dodatkowych danych

Funkcja libssh2_channel_open_ex() zwraca strukturę LIBSSH2_CHANNEL reprezentującą kanał komunikacyjny lub NULL w przypadku błędu.

Plik nagłówkowy <libssh2.h> definiuje makro libssh2_channel_open_session(), które pozwala w wygodny sposób utworzyć kanał typu session:

```
#define LIBSSH2_CHANNEL_WINDOW_DEFAULT 65536
#define LIBSSH2_CHANNEL_PACKET_DEFAULT 32768

#define libssh2_channel_open_session(session) \
libssh2_channel_open_ex((session), "session", sizeof("session") - 1, \
    LIBSSH2_CHANNEL_WINDOW_DEFAULT, \
    LIBSSH2_CHANNEL_PACKET_DEFAULT, NULL, 0)
```

Domyślne wartości dla rozmiaru okna i maksymalnego rozmiaru pakietu nie są dobrane przypadkowo. RFC 4253 wymaga, aby każda implementacja SSH była w stanie odebrać pakiet SSH przenoszący dane o rozmiarze 32768 bajtów lub mniejszym. Domyślny rozmiar okna jest wartością bezpieczną z punktu widzenia wydajności transmisji przez tunel SSH.

Kanał typu session może zostać wykorzystany do wywołania powłoki, polecania systemowego lub podsystemu:

```
int libssh2_channel_shell(LIBSSH2_CHANNEL *channel);

int libssh2_channel_exec(LIBSSH2_CHANNEL *channel,
    const char *message);

int libssh2_channel_subsystem(LIBSSH2_CHANNEL *channel,
    const char *message);
```

Makro libssh2_channel_subsystem() przyjmuje nazwę predefiniowanego podsystemu, np. ”publickey”. Z kolei makro libssh2_channel_exec(), wymaga zdefiniowania polecenia systemowego lub ścieżki do pliku wykonywalnego. Makra zwracają wartość 0 w przypadku powodzenia. Jeżeli makro libssh2_channel_exec() zwróciło wartość, status zdalnego wywołania programu/polecenia można sprawdzić za pomocą funkcji libssh2_channel_get_exit_status():

```
int libssh2_channel_get_exit_status(LIBSSH2_CHANNEL* channel);
```

Funkcja przyjmuje jako jedyny argument strukturę reprezentującą kanał komunikacyjny i zwraca kod wyjścia (status zakończenia) zdalnego procesu.

Operacje wejścia/wyjścia można przeprowadzić za pomocą:

- funkcji libssh2_channel_write_ex(), makra libssh2_channel_write(),
- funkcji libssh2_channel_read_ex(), makra libssh2_channel_read().

Poniżej przedstawiona jest deklaracja funkcji libssh2_channel_write_ex(), która wysyła buflen bajtów korzystając ze strumienia stream_id:

```
ssize_t libssh2_channel_write_ex(LIBSSH2_CHANNEL *channel,
    int stream_id,
    const char *buf,
    size_t buflen);
```

Parametr	Opis
channel	wskaźnik na strukturę reprezentującą kanał komunikacyjny
stream_id	identyfikator strumienia, którym wysyłane będą dane: • stream_id == 0 dane będą transmitowane za pomocą wiadomości SSH_MSG_CHANNEL_DATA • stream_id > 0 dane będą transmitowane za pomocą wiadomości SSH_MSG_CHANNEL_EXTENDED_DATA (podrozdział 1.3.3); stream_id określa typ transmitowanych danych; wartość 1
	jest zarezerwowana dla standardowego wyjścia błędów
buf	wskaźnik na bufor danych do wysłania
buflen	rozmiar danych do wysłania

Funkcja libssh2_channel_write_ex() zwraca liczbę wysłanych bajtów lub wartość ujemną w przypadku błędu.

Do odbierania danych można wykorzystać funkcję libssh2_channel_read_ex():

```
ssize_t libssh2_channel_read_ex(LIBSSH2_CHANNEL *channel,
    int stream_id,
    char *buf,
    size_t buflen);
```

Parametr	Opis
channel	wskaźnik na strukturę reprezentującą kanał komunikacyjny
stream_id	identyfikator strumienia, za pomocą którego dane będą odbierane: • stream_id == 0 dane będą transmitowane za pomocą wiadomości SSH_MSG_CHANNEL_DATA • stream_id > 0 dane będą transmitowane za pomocą wiadomości SSH_MSG_CHANNEL_EXTENDED_DATA (podrozdział 1.3.3); stream_id określa typ transmitowanych danych; wartość 1 jest zarezerwowana dla standardowego wyjścia błędów
buf	wskaźnik na bufor danych, do którego zapisywane będą dane
buflen	rozmiar bufora

Funkcja libssh2_channel_read_ex() zwraca liczbę odebranych bajtów, wartość ujemną w przypadku błędu, wartość 0 – gdy zdalna stacja zamknęła kanał komunikacyjny.

W celu zamknięcia kanału należy wywołać funkcję libssh2_channel_close():

```
int libssh2_channel_close(LIBSSH2_CHANNEL *channel);
```

Funkcja libssh2_channel_close() powoduje wysłanie wiadomości SSH_MSG_CHANNEL_CLOSE, która zamyka jeden kierunek połączenia. Po otrzymaniu tej wiadomości, zdalna strona musi odpowiedzieć wysyłając SSH_MSG_CHANNEL_CLOSE, co zamyka drugi kierunek połączenia. Po wymianie wiadomości, kanał komunikacyjny może zostać zwolniony, a numer kanału ponownie wykorzystany (biblioteka libssh2 nie udostępnia interfejsu, za pomocą którego można by pobrać numeru kanału komunikacyjnego). Za pomocą funkcji libssh2_channel_free() można zwolnić zasoby zaalokowane na rzecz kanału komunikacyjnego:

```
int libssh2_channel_free(LIBSSH2_CHANNEL *channel);
```

1.4. Cel laboratorium

Celem laboratorium jest zapoznanie się z protokołem SSH2 oraz z API biblioteki libssh2. Laboratorium obejmuje aspekty związane z:

- tworzeniem gniazd TCP po stronie klienta,
- uwierzytelnianiem serwera SSH,
- metodami uwierzytelniania użytkowników,
- sesjami protokołu Connection Protocol i zdalnym wywoływaniem poleceń,
- podsystemem sftp (transmisją plików i listowaniem zawartości katalogów).

2. Przebieg laboratorium

Druga część instrukcji zawiera zadania do praktycznej realizacji, które demonstrują zastosowanie technik z omawianego zagadnienia.

2.1. Przygotowanie laboratorium

Do weryfikacji poprawności programów proszę wykorzystać serwer OpenSSH. Konfiguracja serwera powinna umożliwiać uwierzytelnianie użytkowników za pomocą hasła oraz klucza publicznego. Przed przystąpieniem do realizacji zadań, proszę upewnić się, że konfiguracja serwera umożliwia uwierzytelnianie za pomocą wymienionych metod. Plik konfiguracyjny serwera – sshd_config – powinien zawierać następujące opcje:

```
Protocol 2

PasswordAuthentication yes
PermitEmptyPasswords no

PubkeyAuthentication yes
AuthorizedKeysFile .ssh/authorized_keys
```

2.2. Zadanie 1. Sesje Transport Layer Protocol

Zadanie polega na analizie kodu źródłowego przykładowego programu. Program nawiązuje bezpieczne połączenie z serwerem SSH i wyświetla listę metod uwierzytelniających obsługiwanych przez serwer. Argumenty wywołania programu mają następującą postać:

```
authlist [OPCJE] <nazwa użytkownika>@<adres IPv4 lub nazwa hosta>
```

Za pomocą opcji –p, można określić numeru portu, na którym nasłuchuje serwer SSH. Przetwarzanie argumentów wywołania programu oraz nawiązywanie połączenia TCP jest dokonywane za pomocą funkcji zdefiniowanych w pliku libcommon.c. Proszę zapoznać się z jego zawartością, pomijając funkcję odpowiedzialną za analizę argumentów wywołania.

W celu uruchomienia programu należy wykonać następujące czynności:

1. Przejść do katalogu ze źródłami (rozpakować je w razie konieczności).
2. Skompilować program źródłowy do postaci binarnej:

```
$ gcc -c libcommon.c -o libcommon.o
$ gcc authlist.c -lssh2 libcommon.o -o authlist
```

3. Uruchomić snifferfer tshark z następującymi opcjami:

   ```
   $ sudo tshark -i <interfejs> -n -d tcp.port==<port serwera>,ssh \
       port <port serwera>
   ```

   Opcja –V wyświetla szczegóły na temat nagłówków wszystkich protokołów. W przypadku użycia opcji –V najlepiej jest przekierować standardowe wyjście programu do pliku i później przeanalizować jego zawartość (proszę nie używać opcji –w <nazwa pliku>):

   ```
   $ sudo tshark -i <interfejs> -n -d tcp.port==<port serwera>,ssh \
       port <port serwera> -V > "<nazwa pliku>"
   ```

4. Uruchomić program podając nazwę użytkownika oraz adres IP serwera:

   ```
   $ ./authlist <nazwa użytkownika>@<adres IP>
   ```

5. Proszę zaobserwować przebieg komunikacji sieciowej za pomocą sniffera.
6. Czy serwer odpowie listą metod uwierzytelniających, jeżeli podana zostanie fikcyjna nazwa konta użytkownika?

2.3. Zadanie 2. Zdalne wywoływanie poleceń

Celem zadania jest analiza kodu źródłowego przykładowego programu. Program nawiązuje połączenie z serwerem SSH, uwierzytelnia użytkownika za pomocą hasła oraz wywołuje polecenie zdefiniowane w kodzie źródłowym programu. Proszę zwrócić uwagę na sposób w jaki odbierana jest odpowiedź od serwera. Hasło użytkownika pobierane jest w bezpieczny sposób (bez wyświetlania wprowadzanych znaków), za pomocą funkcji zdefiniowanej w pliku libcommon.c. Analiza ciała tej funkcji nie jest wymagana.

W celu uruchomienia programu należy wykonać następujące czynności:

1. Przejść do katalogu ze źródłami.
2. Skompilować program źródłowy do postaci binarnej:

   ```
   $ gcc -c libcommon.c -o libcommon.o
   $ gcc exec.c -lssh2 libcommon.o -o exec
   ```

3. Uruchomić program podając nazwę użytkownika oraz adres IP serwera:

   ```
   $ ./exec <nazwa użytkownika>@<adres IP>
   ```

4. Jaki rezultat daje wywołanie polecenia zdefiniowanego w kodzie źródłowym programu?

2.4. Zadanie 3. Uwierzytelnianie za pomocą klucza publicznego

Proszę zmodyfikować program z zadania 2 w taki sposób, aby uwierzytelnianie użytkownika odbywało się za pomocą metody klucza publicznego. Parę kluczy RSA można wygenerować

za pomocą skryptu keygen.sh lub programu ssh-keygen. Klucze proszę umieścić w katalogu, z którego uruchamiany będzie program. Klucz publiczny należy dodatkowo skopiować na serwer i dodać go do pliku .ssh/authorized_keys w katalogu domowym użytkownika. Klucz publiczny można dodać do listy kluczy autoryzowanych za pomocą polecenia cat:

```
$ cat public.key >> authorized_keys
```

W celu uruchomienia programu należy wykonać następujące czynności:

1. Skompilować program źródłowy do postaci binarnej:

   ```
   $ gcc -c libcommon.c -o libcommon.o
   $ gcc exec_rsa.c -lssh2 libcommon.o -o exec_rsa
   ```

2. Uruchomić program podając nazwę użytkownika oraz adres IP serwera:

   ```
   $ ./exec_rsa <nazwa użytkownika>@<adres IP>
   ```

3. Wprowadzić hasło użyte do zabezpieczenia klucza prywatnego (jeżeli zostało podane podczas generowania kluczy).

2.5. Zadanie 4. Podsystem SFTP

Celem zadania jest analiza kodu programu, który nawiązuje połączenie SSH z serwerem i dokonuje listingu zawartości zdalnego katalogu.

W celu uruchomienia programu należy wykonać następujące czynności:

1. Przejść do katalogu ze źródłami.
2. Skompilować program źródłowy do postaci binarnej:

   ```
   $ gcc -c libcommon.c -o libcommon.o
   $ gcc sftp.c -lssh2 libcommon.o -o sftp
   ```

3. Uruchomić program podając nazwę użytkownika oraz adres IP serwera:

   ```
   $ ./sftp <nazwa użytkownika>@<adres IP>
   ```

4. W razie konieczności, proszę zmodyfikować ścieżkę do katalogu w kodzie źródłowym programu.

2.6. Zadanie 5. Transmisja plików za pomocą SFTP

Proszę zmodyfikować program z zadania 4 w taki sposób, aby po wyświetleniu zawartości katalogu, umożliwiał on pobranie dowolnego pliku. Nazwa pliku ma być wprowadzana przez użytkownika za pomocą klawiatury.

W celu uruchomienia programu należy wykonać następujące czynności:

1. Skompilować program źródłowy do postaci binarnej:

   ```
   $ gcc -c libcommon.c -o libcommon.o
   $ gcc sftp_get.c -lssh2 libcommon.o -o sftp_get
   ```

2. Uruchomić program podając nazwę użytkownika oraz adres IP serwera:

   ```
   $ ./sftp_get <nazwa użytkownika>@<adres IP>
   ```

3. Proszę zweryfikować, czy plik został poprawnie skopiowany.

3. Opracowanie i sprawozdanie

Realizacja laboratorium pt. „SSH” polega na wykonaniu wszystkich zadań programistycznych podanych w drugiej części tej instrukcji. Wynikiem wykonania powinno być sprawozdanie w formie wydruku papierowego dostarczonego na kolejne zajęcia licząc od daty laboratorium, kiedy zadania zostały zadane.

Sprawozdanie powinno zawierać:

- opis metodyki realizacji zadań (system operacyjny, język programowania, biblioteki, itp.),
- algorytmy wykorzystane w zadaniach (zwłaszcza, jeśli zastosowane zostały rozwiązania nietypowe),
- opisy napisanych programów wraz z opcjami,
- trudniejsze kawałki kodu, opisane tak, jak w niniejszej instrukcji,
- uwagi oceniające ćwiczenie: trudne/łatwe, nie/realizowalne, nie/wymagające wcześniejszej znajomości zagadnień (wymienić jakich),
- wskazówki dotyczące ewentualnej poprawy instrukcji celem lepszego zrozumienia sensu oraz treści zadań.