Uniwersytet w Białymstoku

Wydział Matematyczno-Fizyczny

Instytut Matematyki

Jarek Kotowski

Zastosowanie SSL w serwerze Teapop

Praca została napisana
pod kierunkiem
dr. Mariusza Żynela

Białystok 2004

Składam serdeczne podziękowania
dr. Mariuszowi Żynelowi
za wysiłek włożony w pomoc
przy przygotowaniu pracy

Jarek Kotowski

Spis treści

Wstęp

Niemal każdy ma dziś swoją własną skrzynkę pocztową. E-mail stał się jedną z podstawowych form wymiany informacji: pewną, szybką i jeśli o to zadbać również bezpieczną. Z pocztą elektroniczną związane są dwa protokoły SMTP i POP3. Pierwszy z nich SMTP (Simple Mail Transfer Protocol) służy do przesyłania poczty pomiędzy komputerami. Natomiast do odbierania poczty wykorzystywany jest protokół POP3 (Post Office Protocol). Jednak żeby odbierać pocztę potrzebne jest specjalne oprogramowanie zarówno po stronie klienta (programy pocztowe) jak i po stronie serwera. Jedną z implementacji serwera POP3 jest Teapop, który jest tematem mojej pracy. Dokładniej celem mojej pracy było dostosowanie serwera Teapop do obsługi protokołu SSL. Przydatność takiego rozwiązania jest oczywista, jeśli uświadomimy sobie, na jakie zagrożenia jesteśmy narażeni podczas przeglądania, czy pobierania poczty na swój komputer domowy.

Podczas każdej sesji POP3 nazwa użytkownika, jak również hasło są przesyłane w sposób niezaszyfrowany. Przechwycenie tych danych za pomocą *sniffera* nie sprawia problemu nawet początkującemu hakerowi. Od przechwycenia ważnych informacji już tylko krok od nieuprawnionego dostępu do naszej poczty lub, co gorsza do systemu. Chociaż Teapop oferuje szyfrowanie hasła za pomocą polecenia APOP, to cała transmisja jest nadal nieszyfrowana. Na niebezpieczeństwo przechwycenia danych jesteśmy również narażeni ze strony komputerów uczestniczących w transmisji miedzy naszym komputerem a serwerem. Łatwo można się przekonać wykorzystując dowolny program śledzący trasę pakietów, że wymiana danych miedzy klientem a serwerem nie odbywa się bezpośrednio. Uczestniczą w niej inne systemy komputerowe i każdy z nich może uzyskać dostęp do przesyłanych danych. Nie jest to wada programów, ale protokołu POP3, który po prostu nie oferuje szyfrowania połączenia. Jest to związane z tym, że gdy został wymyślony miał służyć do wymiany informacji małej grupie osób i nikt nie przewidywał wówczas tak ogromnego rozwoju Internetu oraz samej poczty elektronicznej. Jednak wykorzystując narzędzia oferowane przez OpenSSL, które zapewnią obsługę protokółu SSL można zabezpieczyć się przed sytuacją podsłuchania danych. Nawet, jeśli transmisja zostania przechwycona to te dane będą bezużyteczne dla osoby, która uzyskała do nich dostęp. Protokół SSL zabezpieczenia nas przed przechwyceniem danych przez osoby nieuprawnione, ale zaletą te-

go protokołu jest potwierdzenie tożsamości serwerów za pomocą certyfikatów cyfrowych. Dzięki temu mamy pewność, że serwer, z którym się połączyliśmy jest rzeczywiście tym, za który się podaje. Tak, więc zastosowanie SSL w serwerze Teapop ma niewątpliwe korzyści. A ponieważ było spore zainteresowanie zmianami, to zmodyfikowana wersja programu została, udostępniona na stronie: http://math.uwb.edu.pl/ mariusz/share/projects/teapop/.

Rozdział 1

Opis protokołu POP3

1.1 Wprowadzenie

POP3, czyli Post Office Protocol, jest jednym z protokołów sieciowych a jego zadanie to odbieranie poczty elektronicznej z serwera, tak by ta trafiła na komputer lokalny. W większości przypadków operacje te są realizowane przez programy pocztowe, np.: Kmail (Linux), Outlook (Windows). Rola użytkownika ogranicza się w tym przypadku do prawidłowej konfiguracji programu pocztowego.

Protokół POP3 został stworzony, aby użytkownicy, których komputery nie pełnią roli serwerów internetowych mogli korzystać z poczty elektronicznej. Poczta trafia na ich konta na serwerach pocztowych przy użyciu protokołu SMTP - wykorzystywanego do przekazywania poczty między serwerami. Aby móc ją pobrać na komputer, wykorzystywany jest protokół POP3. Rola protokołu POP3 ogranicza się wyłącznie do odbierania wiadomości i usuwania ich z serwera. Bardziej zaawansowane i kompleksowe operacje na skrzynkach pocztowych oferuje protokół IMAP4 (RFC1730), którym nie będę się zajmował.

W dalszej części tego rozdziału, termin „klient" będzie odnosił się do komputera korzystającego z usługi POP3, zaś termin "serwer" będzie oznaczać komputer, który udostępnia usługę POP3. Usługa POP3 wykorzystuje port TCP o numerze 110. Kiedy klient życzy sobie skorzystać z protokołu POP3, to nawiązuje połączenie TCP przez port 110 z serwerem. Kiedy połączenie przebiegło pomyślnie, serwer POP3 przesyła pozdrowienia. Teraz klient i serwer POP3 wymieniają komendy i odpowiedzi - kolejno aż połączenie nie zostanie zamknięte lub przerwane.

W czasie połączenia serwer nie rozróżnia wielkości znaków wydawanych komend. Jednak dla większej czytelności warto polecenia pisać wielkimi lite-

rami. Wielkość liter jest istotna przy podawaniu hasła dostępu do skrzynki pocztowej i tylko wtedy. Komendy wysyłane przez klienta mogą być wydane z jednym lub większa liczbą argumentów. Wszystkie rozkazy są kończone przez CRLF (\r\n znak nowego wiersza). Słowa kluczowe i argumenty składają się z drukowalnych znaków ASCII i są oddzielone przez pojedynczy znak spacji (SPACE kod: 32). Słowa kluczowe mają długość trzech, czterech znaków, natomiast argumenty mogą mieć długość do 40 znaków.

Odpowiedzi w POP3 składają się z wskaźnika statusu i słowa kluczowego z następującymi po nim dodatkowymi informacjami. Długość odpowiedzi nie może przekraczać 512 znaków i musi być zakończona przez CRLF. Rozróżniamy dwa identyfikatory stanu:

+OK - dla zdarzeń zakończonych pomyślnie,

-ERR - dla nieudanych operacji.

Serwery muszą wysłać identyfikator stanu +OK i -ERR napisany dużymi literami. Odpowiedzi często zawierają kilka linii. W tym przypadku, po przesłaniu pierwszej linii i znaku CRLF, każde dodatkowo przesłane linie są oddzielone od siebie znakiem nowej linii (CRLF pełniącego rolę separatora). Kiedy cała odpowiedź zostanie przesłana wówczas ostatnia linia składa się z bajtu przerywającego „.” (kropki) oraz CRLF. Jeśli jedna z linii odpowiedzi rozpoczyna się od kropki wówczas, linia ta jest uzupełniana niepotrzebnymi bajtami by zapobiec zakończeniu odpowiedzi w tej linii. W trakcie analizowania takiej odpowiedzi klient sprawdza czy linia nie rozpoczyna się od kropki. Jeśli tak i następnym bajtem nie jest CRLF wówczas pierwszy bajt linii, czyli „.” jest usuwany, a odpowiedź jest przetwarzana dalej. Jeśli po „.” występuje znak CRLF wówczas odpowiedź z serwera POP3 jest kończona i linia zawierająca „.CRLF” nie jest rozważana jako część odpowiedzi.

W czasie pracy z serwerem POP3 można wyróżnić trzy podstawowe stany, są to:

1. Stan autoryzacji (authorization state) - po otworzeniu połączenia TCP i wysłania powitania przez serwer POP3 rozpoczyna się stan autoryzacji. Teraz musimy się „przedstawić” dla serwera POP3, tzn.: podać identyfikator i hasło użytkownika. Jeśli uwierzytelnienie przebiegnie pomyślnie to rozpoczyna się stan transakcji.

2. stan transakcji (transaction state) - użytkownik może wykonywać działania po stronie serwera (przez odpowiednie komendy). Wykonywane są wszystkie operacje związane z przetwarzaniem wiadomości, ich usuwaniem i pobieraniem.

3. stan aktualizacji (update state) - po wpisaniu komendy QUIT przechodzimy do stanu aktualizacji, serwer POP3 zwalnia wszystkie zasoby wykorzystywane podczas stanu transakcji i wysyła pożegnanie. Połączenie TCP jest wtedy zamykane.

Serwer POP3 może przerwać połączenie TCP po pewnym czasie bezczynności (nie wcześniej niż przed upływem dziesięciu minut). Wysłanie dowolnej komendy ze strony klienta powoduje wyzerowanie tego licznika. Po przekroczeniu czasu bezczynności serwer nie przechodzi do stanu aktualizacji (update state), zamyka połączenie bez usuwania wiadomości i nie wysyła tez żadnej odpowiedzi do klienta.

Serwer musi odpowiedzieć na nieprawidłową, niezaimplementowaną komendę negatywnym identyfikatorem statusu (-ERR). Również negatywny identyfikator statusu jest wysłany, kiedy serwer znajduje się w nieprawidłowym stanie pracy. Trudno jest wiec jednoznacznie stwierdzić czy odpowiedz odnosi się do nieprawidłowej komendy czy nierozpoznanego stanu pracy serwera.

1.2 POP3 - Authorization State

Po nawiązaniu przez klienta połączenia z serwerem POP3, są wysyłane pozdrowienia" ze strony serwera. Może to być na przykład:

```
S:  +OK POP3 server ready
```

Teraz sesja POP3 jest w stanie autoryzacji. Klient musi się przedstawić i uwierzytelnić się dla serwera POP3. Są dwa mechanizmy pozwalające na to. Pierwsza metoda polega na wysłaniu kombinacji komend USER oraz PASS,

Polecenia etapu atoryzacji		
Polecenie	**Argument**	**Opis**
USER	nazwa	Nazwa identyfikuje skrzynkę pocztową.
PASS	hasło	Po podaniu nazwy użytkownika trzeba podać hasło jakie ustaliliśmy dla naszej skrzynki.
APOP	nazwa hasło	To alternatywa dla poleceń USER i PASS. Przesyła ona hasło w postaci zaszyfrowanej. Jako drugi argument musimy podać hasło w postaci zaszyfrowanej.
QUIT	-	Komenda ta kończy połączenie.

Tabela 1.1: Polecenia etapu autoryzacji

zaś druga wykorzystuje komendę APOP. Serwer POP3 powinien obsługiwać przynajmniej jedną z tych metod.

1.3 POP3 - Transaction state

Polecenia etapu tranzakcji		
Polecenie	**Argument**	**Opis**
STAT	-	Polecenie to zwraca liczbę wiadomości zapisanych na serwerze i łączny rozmiar wszystkich wiadomości w bajtach.
LIST	liczba	Polecenie zwraca rozmiar wiadomości o podanym numerze, lub wszystkich jeśli nie podamy argumentu. Odpowiedź serwera to „+OK x b", gdzie x to podany numer wiadomości, b - rozmiar wiadomości w bajtach.
RETR	liczba	W ten sposób odczytujemy wiadomość o numerze podanym jako argument. Jeśli nie wystąpi błąd to otrzymamy odpowiedź postaci: „+OK b", gdzie b to rozmiar wiadomości w bajtach, zaś kolejne linie będą zawierały treść wiadomości.
DELE	liczba	Tym poleceniem zaznaczamy wiadomość do usunięcia. Wiadomość nie zostanie od razu usunięta, jej fizyczne skasowanie nastąpi dopiero po wydaniu polecenia QUIT kończącego połączenie.
RSET	-	Polecenie to odznacza wiadomości, które zostały zaznaczone do usunięcia.
NOOP	-	To polecenie, zawsze daje odpowiedź pozytywną. Służy do utrzymania połączenia z serwerem.

Tabela 1.2: Polecenia etapu transakcji

Po tym jak klient szczęśliwie zidentyfikował się dla serwera POP3, sesja POP3 jest w stanie transakcji. Klient może wielokrotnie wydawać dowolną z poniższych komend. Ostatecznie stan transakcji kończy się, gdy klient wykorzysta komendę QUIT, sesja POP3 przechodzi w stan aktualizacji.

1.4 POP3 - Update state

Polecenia etapu aktualizacji		
Polecenie	**Argument**	**Opis**
QUIT	-	Komenda ta kończy połączenie TCP i powoduje wykonanie wszystkich zmian.

Tabela 1.3: Polecenia etapu aktualizacji

1.5 Podsłuchanie transmisji POP3

Transmisja pomiędzy serwerem poczty a klientem pocztowym nie jest w żaden sposób szyfrowana, w szczególności hasło użytkownika przekazywane jest otwartym tekstem. Nawet korzystając z polecenia APOP dostępnego w POP3, uchronimy jedynie nasze hasło i nazwę użytkownika, ale nasza korespondencja nadal pozostanie jawna dla osoby podsłuchującej.

1.5.1 Programy podsłuchujące

Sniffer lub *packet sniffer* jest programem przechwytującym ruch w sieci komputerowej. Programy te przechwytują pakiety przy użyciu określonego interfejsu lub wszystkich interfejsów systemu. Jest to możliwe, ponieważ *ethernet* jest rodziną protokołów, o logicznej topologii magistrali. Oznacza to, że informacja wysłana przez nadawcę trafia do wszystkich komputerów w sieci lokalnej, przy czym odczytuje ją jedynie odbiorca, którego *adres MAC karty sieciowej* jest zgodny z adresem przeznaczenia. Po przestawieniu karty sieciowej w tryb *promiscous*, karta ignoruje adres odbiorcy i przetwarza wszystkie pakiety nawet te, które nie są adresowane do niej.

Wykorzystując programy podsłuchujące można podsłuchać transmisję pomiędzy określonymi hostami w sieci, a także można podsłuchiwać na określonym porcie. Narzędzia tego typu pomagają nie tylko hakerom, ale przede wszystkim administratorom sieci w rozwiązywaniu problemów, takich jak wąskie gardła lub spadek wydajności. Można również potwierdzić atak na naszą sieć, dzięki czemu administrator może wprowadzić do konfiguracji sieci zmiany zapewniające jej wydajne i bezpieczne funkcjonowanie. Wykorzystałem następujące programy do przechwytywania pakietów:

- **Tcpdump.** Narzędzie monitorujące ruch w sieci, uruchamiane z wiersza poleceń Istnieje ono już od dłuższego czasu i oparta jest na nim większość narzędzi posiadających interfejs graficzny. Tcpdump, znajduje się pod adresem http://www.tcpdump.org/

- **Snoop.** Narzędzie monitorujące ruch w sieci, uruchamiane jak Tcpdump z wiersza poleceń. Posługiwałem się tym narzędziem ze względu na system operacyjny, na którym testowane było działanie serwera Teapop.
- **Ethereal.** Narzędzie monitorujące ruch w sieci, posiadające interfejs graficzny znacznie bardziej przyjazny dla użytkownika niż Tcpdump czy Snoop. Pozwala na oglądanie wyników w czasie rzeczywistym, oraz na interaktywne przeglądanie każdego z przechwyconych pakietów i jego nagłówków. Strona domowa programu Ethereal znajduje się pod adresem: http://www.ethereal.com/

1.5.2 Co w sieci piszczy?

Do podsłuchania transmisji użyłem narzędzia konsolowego o nazwie *tcpdump*, jest to program przeznaczony dla systemów z rodziny Unix/Linux. Przykładowe wywołanie programu *tcpdump* może wyglądać:

```
tcpdump   [ -aenqtvxX ] [ -c liczba ] [ -F plik ]
          [ -i interfejs ] [ -s liczba ] [ -w plik]
          [ wyrażenie ]
```

-a - program próbuje przekształcić adresy sieci i adresy rozgłoszeniowe na nazwy

-c liczba - kończy pracę programu po przechwyceniu *liczba* pakietów

-e - dla każdego przechwyconego pakietu wyświetla nagłówek na poziomie łącza

-F - dla wyrażeń filtra zamiast opcji i wyrażeń można wykorzystać w charakterze danych plik, wówczas wyrażenie podane w linii poleceń będzie zignorowane

-i - określa interfejs dla programu Tcpdump. Jeśli interfejs nie został określony, program poszukuje najniższego numeru interfejsu, z wyjątkiem interfejsu *loopback*, a następnie wyświetla pakiety pochodzące z tego interfejsu

-n - nie wyświetla nazw serwerów, prezentując jedynie ich adresy w postaci liczbowej

-q - szybkie dane wyjściowe, w celu ograniczenia ilości informacji o protokole

-s - liczba - pobiera określoną przez liczba ilość bajtów z każdego przechwyconego pakietu. Wartość 0 oznacza ze wszystkie bajty z pakietu będą wyświetlane.

-t - nie wyświetla znacznika czasu (ang. timestamp)

-w - zapisuje przechwycone pakiety w pliku, zamiast wypisywać je na ekran

-v - wyświetla dane wyjściowe w rozszerzonej formie

-x - wyświetla dane w postaci szesnastkowej

-X - gdy dane są wyświetlane w postaci szesnastkowej, to także są wyświetlane w kodzie ASCII

wyrażenie - określa filtr pakietów, które mają być przechwytywane

Rysunek 1.1: Schemat sieci użytej do podsłuchania transmisji POP3

Schemat użytej sieci został przedstawiony na rysunku 1.1. Komputery w tej sieci pracowały pod kontrolą systemu operacyjnego Solaris9, dodatkowo na komputerze 192.168.2.104 został uruchomiony serwer Teapop-0.3.8, a na komputerze 192.168.2.105 zainstalowano pakiet tcpdump. Rolę klienta pełnił komputer o adresie IP 192.168.2.104, który odbierał pocztę z komputera 192.168.2.104 za pomocą programu pocztowego wbudowanego w przeglądarkę internetową Mozilla. Na komputerze o adresie logicznym 192.168.2.105 uruchomiono program tcpdump z parametrami:

```
tcpdump -s0 -x -X -i eth0 host 192.168.2.104 port 110
```

W rezultacie otrzymano następujące pakiety:

```
192.168.2.104 - 192.168.2.108 TCP D=110 S=1153 Syn Seq=2663134520 Len=0
Win=16384 Options=<mss 1460,nop,nop,sackOK>

          0: 0010 4bd2 7fc6 0010 4b6d df45 0800 4500    ..K.....Km.E..E.
         16: 0030 2758 4000 8006 4d4b c0a8 0268 c0a8    .0'X@...MK...h..
         32: 026c 0481 006e 9ebc 3538 0000 0000 7002    .l...n..58....p.
```

```
         48: 4000 e416 0000 0204 05b4 0101 0402        @.............

192.168.2.108 - 192.168.2.104 TCP D=1153 S=110 Syn Ack=2663134521 Seq=1896522730 Len=0
Win=64240 Options=<mss 1460,nop,nop,sackOK>

          0: 0010 4b6d df45 0010 4bd2 7fc6 0800 4500    ..Km.E..K.....E.
         16: 0030 4037 4000 4006 746c c0a8 026c c0a8    .0@7@.@.tl...l..
         32: 0268 006e 0481 710a a3ea 9ebc 3539 7012    .h.n..q.....59p.
         48: faf0 1420 0000 0204 05b4 0101 0402         ... ..........

192.168.2.104 - 192.168.2.108 TCP D=110 S=1153 Ack=1896522731 Seq=2663134521 Len=0
Win=17520

          0: 0010 4bd2 7fc6 0010 4b6d df45 0800 4500    ..K.....Km.E..E.
         16: 0028 2759 4000 8006 4d52 c0a8 0268 c0a8    .('Y@...MR...h..
         32: 026c 0481 006e 9ebc 3539 710a a3eb 5010    .l...n..59q...P.
         48: 4470 f764 0000 0000 0000 0000              Dp÷d........
```

Te trzy pakiety służą do nawiązania połączenia z serwerem. Więcej o uzgadnianiu i kończeniu połączenia TCP można znaleźć w dodatku A, na stronie 46. Teraz, gdy połączenie zostało pomyślnie nawiązane serwer wysyła komunikat powitalny:

```
192.168.2.108 - 192.168.2.104 TCP D=1153 S=110 Push Ack=2663134521 Seq=1896522731 Len=82
Win=64240

          0: 0010 4b6d df45 0010 4bd2 7fc6 0800 4500    ..Km.E..K.....E.
         16: 007a 4038 4000 4006 7421 c0a8 026c c0a8    .z@8@.@.t!...l..
         32: 0268 006e 0481 710a a3eb 9ebc 3539 5018    .h.n..q.....59P.
         48: faf0 1ef1 0000 2b4f 4b20 5465 6170 6f70    ......+OK Teapop
         64: 205b 302e 332e 385d 202d 2054 6561 7370     [0.3.8] - Teasp
         80: 6f6f 6e20 7374 6972 7320 6172 6f75 6e64    oon stirs around
         96: 2061 6761 696e 203c 3130 3831 3936 3134     again <10819614
        112: 3639 2e36 3731 3538 3934 4140 4c6c 7977    69.6715894A@Llyw
        128: 656c 6c79 6e3e 0d0a                        ellyn>..
```

Komputer 192.168.2.104 musi przejść przez stan autoryzacji, a więc podać nazwę użytkownika i hasło by uzyskać dostęp do skrzynki pocztowej. Jako pierwsza jest przesyłana nazwa użytkownika:

```
192.168.2.104 - 192.168.2.108 TCP D=110 S=1153 Push Ack=1896522813 Seq=2663134521 Len=13
Win=17438

          0: 0010 4bd2 7fc6 0010 4b6d df45 0800 4500    ..K.....Km.E..E.
         16: 0035 2760 4000 8006 4d3e c0a8 0268 c0a8    .5'`@...M>...h..
         32: 026c 0481 006e 9ebc 3539 710a a43d 5018    .l...n..59q..=P.
         48: 441e ec57 0000 5553 4552 2074 6561 706f    D..W..USER teapo
         64: 700d 0a                                    p..
```

Nazwa użytkownika występuje po komendzie USER i jest od niej oddzielona pojedynczą spacją. Odczytując nazwę użytkownika należy pominąć dwa ostatnie bajty wiadomości, ponieważ jest to znak CRLF kończący każdy rozkaz wysłany do serwera POP3. Ciężko jest to stwierdzić patrząc jedynie na reprezentację pakietu w kodzie ASCII, ponieważ są to dwie kropki. Jednak informacje o pakiecie są wyświetlane również szesnastko, a dwa ostatnie bajty tej wiadomości to: `0d 0a`, które mówią, że jest to znak CRLF. Tak, więc nazwa użytkownika przesłana do serwera to: `teapop`.

```
192.168.2.108 - 192.168.2.104 TCP D=1153 S=110 Ack=2663134534 Seq=1896522813 Len=0
Win=64240

          0: 0010 4b6d df45 0010 4bd2 7fc6 0800 4500    ..Km.E..K.....E.
         16: 0028 4039 4000 4006 7472 c0a8 026c c0a8    .(@9@.@.tr...l..
         32: 0268 006e 0481 710a a43d 9ebc 3546 5010    .h.n..q..=..5FP.
         48: faf0 4085 0000 0000 0000 0000              ..@.........

192.168.2.108 - 192.168.2.104 TCP D=1153 S=110 Push Ack=2663134534 Seq=1896522813 Len=42
Win=64240

          0: 0010 4b6d df45 0010 4bd2 7fc6 0800 4500    ..Km.E..K.....E.
         16: 0052 403a 4000 4006 7447 c0a8 026c c0a8    .R@:@.@.tG...l..
         32: 0268 006e 0481 710a a43d 9ebc 3546 5018    .h.n..q..=..5FP.
         48: faf0 65cd 0000 2b4f 4b20 5765 6c63 6f6d    ..e...+OK Welcom
         64: 652c 2064 6f20 796f 7520 6861 7665 2061    e, do you have a
         80: 6e79 2074 7970 6520 6f66 2049 443f 0d0a    ny type of ID?..
```

Nazwa użytkownika została zaakceptowana przez serwer (192.168.2.108) o czym świadczy komunikat: `+OK Welcome, do you have any type of ID`. Teraz serwer oczekuje na hasło, które niezwłocznie jest wysłane przez program pocztowy:

```
192.168.2.104 - 192.168.2.108 TCP D=110 S=1153 Push Ack=1896522855 Seq=2663134534 Len=13
Win=17396

          0: 0010 4bd2 7fc6 0010 4b6d df45 0800 4500    ..K.....Km.E..E.
         16: 0035 2763 4000 8006 4d3b c0a8 0268 c0a8    .5'c@...M;...h..
         32: 026c 0481 006e 9ebc 3546 710a a467 5018    .l...n..5Fq..gP.
         48: 43f4 32da 0000 5041 5353 2031 3233 6162    C.2...PASS 123ab
         64: 630d 0a                                    c..
```

Jak widać także hasło jest przesyłane w postaci nie zaszyfrowanej, więc nie ma większych problemów z jego odczytaniem. Wiedząc, że hasło występuje po komendzie PASS i jest od niej oddzielone pojedynczą spacją, można je odczytać i jest to ciąg znaków: `123abc`. Podobnie jak przy odczytywaniu nazwy użytkownika należy pominąć dwa ostatnie bajty.

```
192.168.2.108 - 192.168.2.104 TCP D=1153 S=110 Ack=2663134547 Seq=1896522855 Len=0
Win=64240

          0: 0010 4b6d df45 0010 4bd2 7fc6 0800 4500    ..Km.E..K.....E.
         16: 0028 403b 4000 4006 7470 c0a8 026c c0a8    .(@;@.@.tp...l..
         32: 0268 006e 0481 710a a467 9ebc 3553 5010    .h.n..q..g..5SP.
         48: faf0 404e 0000 0000 0000 0000              ..@N........

192.168.2.108 - 192.168.2.104 TCP D=1153 S=110 Push Ack=2663134547 Seq=1896522855 Len=37
Win=64240

          0: 0010 4b6d df45 0010 4bd2 7fc6 0800 4500    ..Km.E..K.....E.
         16: 004d 403c 4000 4006 744a c0a8 026c c0a8    .M@<@.@.tJ...l..
         32: 0268 006e 0481 710a a467 9ebc 3553 5018    .h.n..q..g..5SP.
         48: faf0 3b61 0000 2b4f 4b20 4927 6d20 7265    ..;a..+OK I'm re
         64: 6164 7920 746f 2073 6572 7665 2079 6f75    ady to serve you
         80: 2c20 4d61 7374 6572 2e0d 0a                , Master...
```

Hasło zostało zaakceptowane przez serwer, o czym informuje nas komunikat: `+OK I'm ready to serve you, Master` i host 192.168.2.104 przeszedł do stanu transakcji, może wykonywać różnego rodzaju operacje na skrzynce

pocztowej za pomocą dostępnych komend. Aby wykonane na serwerze operacje odniosły skutek musi zostać wydane polecenie QUIT.

```
192.168.2.104 - 192.168.2.108 TCP D=110 S=1153 Push Ack=1896524185 Seq=2663134592 Len=6
Win=16066

           0: 0010 4bd2 7fc6 0010 4b6d df45 0800 4500    ..K.....Km.E..E.
          16: 002e 277c 4000 8006 4d29 c0a8 0268 c0a8    ..'|@...M)...h..
          32: 026c 0481 006e 9ebc 3580 710a a999 5018    .l...n..5.q...P.
          48: 3ec2 4f5c 0000 5155 4954 0d0a              >.O\..QUIT..

192.168.2.108 - 192.168.2.104 TCP D=1153 S=110 Push Ack=2663134598 Seq=1896524185 Len=41
Win=64240

           0: 0010 4b6d df45 0010 4bd2 7fc6 0800 4500    ..Km.E..K.....E.
          16: 0051 4047 4000 4006 743b c0a8 026c c0a8    .Q@G@.@.t;...l..
          32: 0268 006e 0481 710a a999 9ebc 3586 5018    .h.n..q.....5.P.
          48: faf0 b595 0000 2b4f 4b20 4974 2068 6173    ......+OK It has
          64: 2062 6565 6e20 6120 706c 6561 7375 7265     been a pleasure
          80: 2073 6572 7669 6e67 2079 6f75 2e0d 0a       serving you...
```

Po otrzymaniu komendy QUIT, serwer kończy połączenie i zatwierdza zmiany dokonane przez klienta. Transmisja została zakończona pomyślnie, przynajmniej teoretycznie, bo komputery 192.168.2.104 i 192.168.2.108 nie zdają sobie sprawy, że oprócz nich w połączeniu brał udział komputer 192.168.2.105. Wie on już, jakie hasło do skrzynki pocztowej ma użytkownik `teapop`. Jak można było się przekonać nazwa użytkownika i hasło są przesyłane w postaci jawnej, a co najważniejsze nie ma żadnego kłopotu z ich odczytaniem dysponując odpowiednim oprogramowaniem.

Rozdział 2

Opis SSL

Celem niniejszego rozdziału jest przedstawienie protokołu SSL wraz z leżącymi u jego podstaw mechanizmami kryptograficznymi oraz praktycznych aspektów jego zastosowania. Protokół TCP/IP, jak również protokoły na nim oparte takie jak: HTTP i POP3 nie umożliwiają szyfrowania czy ochrony indywidualnych połączeń. Dzięki protokołowi SSL użytkownicy i serwery mogą chronić dane przekazywane w Internecie. Niezabezpieczone dane przesyłane w Internecie mogą zostać sfałszowane lub wykorzystane przez innych użytkowników tej sieci. W normalnych warunkach informacje są przesyłane między komputerem a serwerem w procesie zwanym skokiem IP, który może obejmować wiele systemów komputerowych. Każdy z tych systemów stanowi potencjalne zagrożenie dla bezpieczeństwa transmitowanych danych, ponieważ może uzyskać do nich dostęp. Aby uniemożliwić hakerom i komputerom pośredniczącym w transmisji, przechwytywanie poufnych danych lub zakłócanie transmisji, należy skorzystać z szyfrowania.

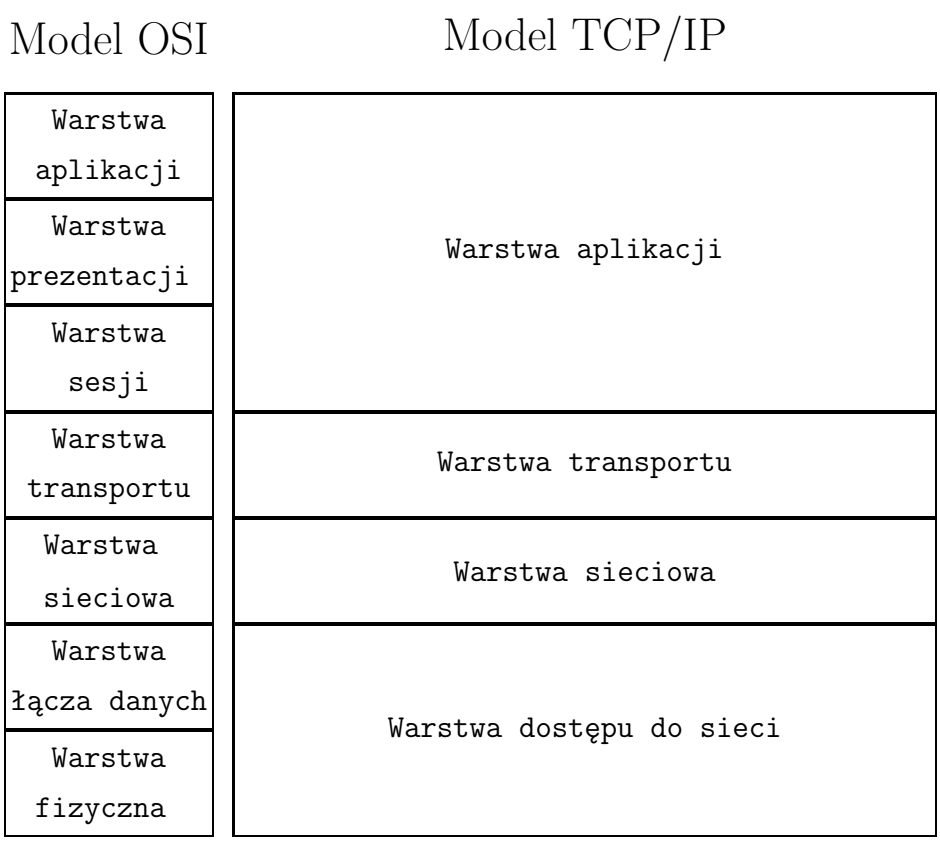
2.1 Protokół SSL

Protokół SSL (Secure Sockets Layer) został opracowany pod koniec lat osiemdziesiątych przez firmę Netscape w celu usprawnienia bezpiecznej komunikacji między serwerami internetowymi a przeglądarkami. Niebawem jednak stał się nieoficjalnym standardem, stosowanym w Internecie do przesyłania poufnych informacji. O powszechności protokołu SSL może świadczyć fakt ze zarówno wersja 2 jak i wersja 3 protokołu SSL są wbudowane w niemal wszystkie serwery WWW lub stanowią odrębny moduł (np. moduł serwera Apache). Protokół SSL obecnie jest dostępny we wszystkich ważniejszych przeglądarkach WWW. W oparciu o trzecią wersję protokołu (SSLv3) jest tworzony obecnie ulepszony protokół TLS (Transport Layer Security).

Do głównych zalet i podstawowych funkcji tego protokołu typu klient-serwer należy:

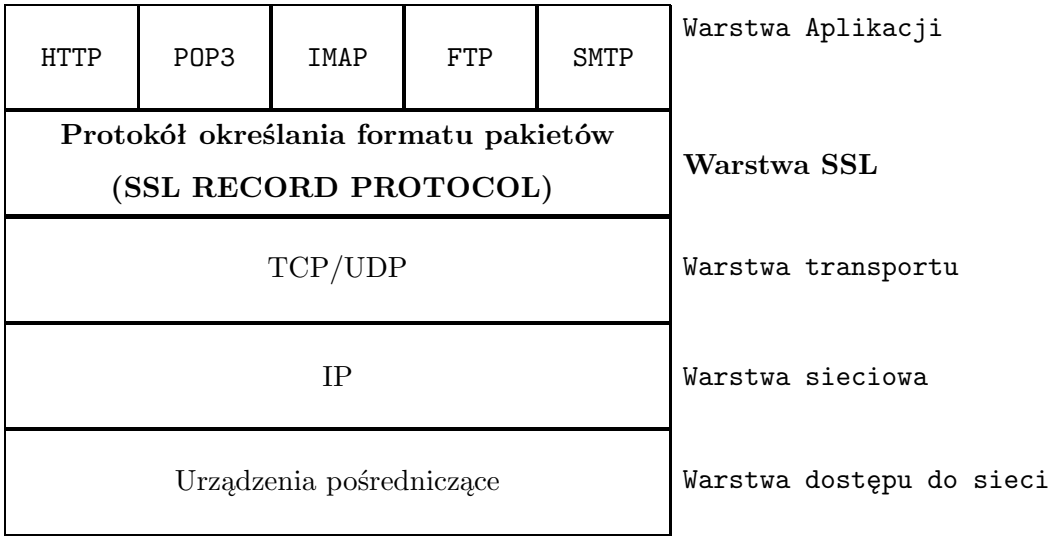
- Uwierzytelnienie - czyli potwierdzanie tożsamości serwerów za pomocą certyfikatów cyfrowych (ewentualnie również użytkownika).
- Poufność - ochrona przesyłanych danych przed podsłuchem przez ich zaszyfrowanie.
- Integralność - gwarantuje, że wysłane informacje dotrą do adresata nie zmodyfikowane i nieodczytane przez nikogo po drodze.

2.2 SSL w modelu ISO/OSI



Rysunek 2.1: Cztero warstwowy model TCP/IP w kontekście modelu OSI. Poszczególne warstwy protokołów TCP/IP(oprócz transportowego i sieciowego) nie odpowiadają warstwom w modelu OSI/ISO. Warstwa fizyczna oraz łącza danych tworzą tu warstwę dostępu do sieci, w skład warstwy aplikacji wchodzi również warstwa prezentacji oraz sesji

Model TCP/IP ma budowę czterowarstwową. Wysyłanie danych odbywa się od warstwy najwyższej do warstwy najniższej. Dla komputera wysyłającego



Rysunek 2.2: Protokół SSL dodaje kolejną warstwę do stosu protokołu odpowiedzialną za działanie SSL

dane transmisja odbywa się od warstwy aplikacji poprzez warstwę transportową, internetu aż trafia do warstwy sieci, gdzie dane są wysyłane fizycznie. W komputerze przyjmującym, dane przebywają trasę odwrotną - poszczególne warstwy odklejają kolejne nagłówki tak żeby do aplikacji odbiorcy dotarły dane z programu nadawcy. Oprócz modelu TCP/IP istnieje jeszcze model OSI (Open System Connection), został opracowany przez organizację ISO i jest standardem komunikacji sieciowej, składającym się z siedmiu współpracujących ze sobą warstw protokołów sieciowych.

W internetowym stosie przetwarzania protokół SSL jest umieszczany powyżej warstwy transportowej (obejmującej moduł TCP - Transport Control Protocol) i poniżej warstwy aplikacji zawierającej protokoły HTTP, SMTP, Telnet, FTP, Gopher i NNTP. Umieszczenie warstwy SSL pod warstwą aplikacji, a nad warstwami protokołu TCP/IP, nie wpływa zasadniczo na funkcjonowanie protokołów innych warstw. Pozwala to na użycie go w innych aplikacjach opartych na technikach internetowych.

2.3 Kryptografia w SSL

2.3.1 Szyfry symetryczne

W szyfrowaniu symetrycznym zarówno nadawca szyfrujący wiadomości jak odbiorca dekodujący dane używa tego samego klucza. Jednym z warunków

tego czy transakcja będzie przebiegać w sposób bezpieczny zależy od tego czy klucz od momentu zaszyfrowania informacji do momentu jej odszyfrowania będzie tajny. Jego główną zaletą jest to, iż proces kodowania i dekodowania jest dokonywany o wiele szybciej niż w przypadku metody asymetrycznej - wiąże się to bezpośrednio z długością klucza. Stosuje się, klucze o długości:

- 40 bitowy to zbyt małe bezpieczeństwo,
- 56 bitowy jest stosunkowo bezpieczny,
- 128 bitowy jest bardzo bezpieczny.

Algorytm DES

Algorytm DES (Data Encryption Standard) został opracowany w latach 70-tych XX wieku przez firmę IBM w oparciu o algorytm Lucifier Horsta Feistela. W 1977 roku Narodowe Biuro Standardów USA zaakceptowało ten algorytm jako standard szyfrowania danych nieutajnionych przez urzędy państwowe. Obecnie DES jest bardzo rozpowszechnionym algorytmem, ale niedającym już odpowiedniego bezpieczeństwa szyfrowanych danych. Zasada działania w tym algorytmie jest następująca: 64 bitowe bloki danych są szyfrowane 56-bitowym kluczem. Dopiero szyfrowanie kluczem 128 bitowy uważa się za bezpieczne. Nowszą odmianą tego algorytmu jest 3DES, bazujący na sekwencji szyfruj-deszyfruj-szyfruj z trzema niepowiązanymi ze sobą kluczami - potrójny DES. Zaletą 3DES jest jego szybkość działania.

Algorytm IDEA

Innym znanym algorytmem symetrycznym jest algorytm IDEA (International Data Encryption Algorithm). Został on wynaleziony na początku lat 90-tych przez Xuejia Lai i Jamesa Masseya na ETH w Zurychu w Szwajcarii. Stosuje on 128-bitowy klucz i jest uważany za bezpieczny. Podczas próby metodą siłową należałoby wykonać 1038 prób, co przy dzisiejszym stanie technologii jest praktycznie nie wykonalne. Obecnie jest to jeden z najbardziej znanych algorytmów publicznych. Jest wykorzystywany miedzy innymi w PGP - oprogramowanie do szyfrowania poczty elektronicznej.

2.3.2 Szyfry niesymetryczne

Szyfry asymetryczne nazywane też szyfrowaniem z kluczem publicznym. Metoda ta polega na tym, iż do szyfrowania danych używamy jednego klucza (najczęściej jest to klucz publiczny), a do deszyfrowania tego, co zostało już zakodowane drugiego (najczęściej jest to klucz prywatny). Klucz prywatny jak i publiczny to nic innego jak unikalne ciągi bitów, im dłuższe tym istnieje

mniejsze prawdopodobieństwo ich złamania. Klucz prywatny wraz z publicznym stanowią parę i tylko zastosowanie kluczy ze sobą powiązanych zapewnia prawidłową transmisję. Rozwiązanie takie pozwala na wyeliminowanie tzw. strony zaufanej, do której należy przekazywanie klucza jak ma to miejsce w przypadku szyfrowania symetrycznego. Klucz prywatny powinien znajdować się na serwerze i nigdy nie powinien go opuszczać, natomiast klucz publiczny wraz z certyfikatem udostępniany jest każdemu, kto próbuje wysłać dane w bezpieczny sposób. Osoba ta szyfruje dane używając klucza publicznego, a deszyfracji tej wiadomości można dokonać tylko za pomocą klucza prywatnego. Obecnie przyjmuje się, że klucz:

- 512 bitowy to zbyt małe bezpieczeństwo,
- 768 bitowy jest stosunkowo bezpieczny,
- 1024 bitowy zapewnia właściwe bezpieczeństwo.

Algorytm RSA

Algorytm RSA (Rivest Shamir Adleman) został opracowany w roku 1977 przez trzech matematyków, profesorów z MIT (Massachussets Institute of Technology) L.R. Rivesta, A. Shamira i L.M. Adlemana. Jego nazywa utworzona jest od pierwszych liter nazwisk autorów: w skrócie RSA. Jest dobrze znanym i szeroko stosowanym algorytmem z kluczem publicznym. Bezpieczeństwo RSA opiera się na trudności rozkładu dużych liczb na iloczyny liczb pierwszych. O stopniu trudności złamania szyfru może świadczyć fakt, ze umieszczona w roku 1977 przez autorów algorytmu zagadka w czasopiśmie „Scientific American", została rozwiązana dopiero w 1994 roku. Polegała ona na odtworzenie tekstu z szyfrogramu. Do stworzenia szyfrogramu został wykorzystany szyfr RSA-129, liczba 129 w oznaczeniu bierze się od 129 cyfrowej liczby, którą trzeba rozłożyć na czynniki pierwsze by uzyskać klucz deszyfrujący i funkcję odwrotną deszyfrującą szyfrogram. Szyfr RSA-129 został w końcu złamany w 1994 roku dzięki równoległej pracy wielu komputerów w sieci komputerowej i superkomputerów równoległych.

Obecnie algorytm RSA uważa się za bezpieczny, jeśli się używa odpowiednio długich kluczy. 512 bitów to zbyt mało, na złamanie takiego szyfru korzystając z opracowanej w 1999 r. przez Adi Shamira metody wystarczą 2 dni. Klucze 1024, 2048 bitowe zapewniające właściwe bezpieczeństwo tzn. są nie do złamania przy współczesnym stanie wiedzy o algorytmach komputerowych i przy współczesnych możliwościach obliczeniowych systemów komputerowych.

Osoba biorąca udział w wymianie informacji dysponuje dwoma kluczami prywatnym (utajnionym) oraz publicznym. Podstawowe przeznaczenie klucza

prywatnego to deszyfrowanie wiadomości lub składanie podpisu cyfrowego, zaś klucza publicznego: szyfrowanie wiadomości dla odbiorcy lub sprawdzanie jego tożsamości na podstawie podpisu. Klucze te mają następującą własność, jeśli coś zostanie zaszyfrowane jednym kluczem to odszyfrować to można tylko wykorzystując do tego drugi klucz. Poza tym nie da się wywnioskować na podstawie klucza publicznego budowy klucza prywatnego i odwrotnie.

Algorytm RSA został przez autorów opatentowany. Patent opiewa na Massachussets Institute of Technology i obejmuje zarówno szyfrowanie jak i podpisy cyfrowe.

2.3.3 Funkcje haszujące

Funkcje haszujące są funkcjami jednokierunkowymi, zamieniającymi treść wiadomości na liczbę w ten sposób, że żadne dwie różne wiadomości nie dadzą tej samej liczby wynikowej oraz nie ma możliwości odtworzenia oryginalnej wiadomości na podstawie otrzymanej liczby. Najpopularniejszymi obecnie algorytmami do tworzenia skrótów wiadomości są MD5 (Message Digest 5) oraz SHA (Secure Hash Algorithm). MD5 zamienia dowolną wiadomość na liczbę 128-bitową, zaś SHA na liczbę 160-bitową. Algorytm MD5 znalazł szerokie zastosowanie w internecie do zapewnienia nie zmienności udostępnianych plików. Chodzi tutaj o uniemożliwienie zmiany kodu pliku przez nieuprawnione osoby i podmienieniu go na przykład na konia trojańskiego. W praktyce działa to w następujący sposób, w raz z aplikacją możemy pobrać jej skrót, który powinien być przechowywany w bazie danych lub też na innym serwerze plików. Po pobraniu tych plików powinniśmy sami obliczyć sumę kontrolną programu i porównać go z oryginałem, jeśli oba się zgadzają to znaczy, że plik nie został naruszony.

2.4 Działanie protokołu

Aby uzyskać jak najwyższy poziom bezpieczeństwa w SSL są wykorzystywane szyfry symetryczne jak niesymetryczne. Do ustanowienia połączenia wykorzystywane jest szyfrowanie asymetrycznej, natomiast bezpieczeństwo wymiany danych zapewnia szyfrowanie symetryczne. Dzieje się tak z następujących powodów: po pierwsze, algorytmy z kluczem publicznym są znacznie wolniejsze od szyfrów symetrycznych, przez co nie nadają się do wydajnego szyfrowania danych przesyłanych w dużych ilościach. Z drugiej jednak strony zastosowanie tylko i wyłącznie szyfru symetrycznego wymagałoby bezpiecznej wymiany klucza szyfrującego. Połączenie dwóch rodzin szyfrów rozwiązuje oba te problemy w sposób efektywny i elegancki.

2.4.1 Ustanowienie połączenia SSL

W momencie nawiązania przez klienta połączenia z serwerem, serwer wysyła swój certyfikat. Klient podejmuje decyzję czy wierzy on w tożsamość serwera. Ponieważ certyfikat jakiegokolwiek serwera jest podpisany zawsze kluczem prywatnym instytucji certyfikującej (CA), która zaświadcza o jego tożsamości, aplikacja wyszukuje w swych zasobach certyfikatu tej właśnie instytucji. W przypadku, gdy go nie znajdzie użytkownik jest informowany o tym i bezpieczne połączenie nie zostanie ustanowione bez jego zgody. Natomiast, jeżeli taki certyfikat posiada to również posiada klucz publiczny tej instytucji, a więc ma możliwość odczytania certyfikatu serwera. W tym momencie na podstawie danych zawartych w certyfikacie następuje ustalenie algorytmu kodującego i wygenerowanie losowego klucza symetrycznego sesji. Klucz ten jest przesyłany do serwera w postaci zaszyfrowanej otrzymanym kluczem publicznym. Dalsza wymiana danych między serwerem a klientem będzie szyfrowana jednym z algorytmów symetrycznych przy użyciu klucza sesji. Proces ten nazywamy „handshaking'iem". Plan nawiązania połączenia szyfrowanego SSL jest przedstawiony poniżej:

1. Nawiązanie połączenia z serwerem.
2. Autoryzacja serwera wobec klienta.
3. Wybranie algorytmów szyfrowania i poziomu bezpieczeństwa.
4. Opcjonalnie autoryzacja klienta wobec serwera. Wygenerowanie w sposób losowy kluczy sesji za pomocą kluczy publicznych, które później będą użyte do transmisji danych.
5. Ustalenie połączenia SSL.

2.4.2 Wymiana danych

Wygenerowanie losowego ciągu danych, deszyfracja certyfikatów cyfrowych, generowanie kluczy i przeprowadzenie szyfrowania kluczem publicznym wymaga czasu, a więc procedura uzgadniania trwa stosunkowo długo. Na szczęście wyniki są zapisywane w pamięci podręcznej i nie trzeba przeprowadzać uzgadniania przed wysłaniem kolejnych danych. Do szyfrowania właściwych danych w SSL jest wykorzystywany algorytm symetryczny, jeden z uzgodnionych podczas początkowej wymiany. Klucz szyfrujący jest generowany losowo dla każdego połączenia.

W SSL w trakcie przesyłania zaszyfrowanych danych podejmowane są następujące akcje:

1. Dane dzielone są na małe, możliwe do użycia partie.

2. Każdy pakiet może, (choć nie musi) zostać skompresowany.

3. Za pomocą odpowiedniego algorytmu skrótu dla każdego pakietu obliczany jest kod uwierzytelniający.

4. Skompresowane dane i ich kod uwierzytelniający zostają połączone i zaszyfrowane.

5. Zaszyfrowane pakiety wraz z dołączonymi do nich nagłówkami informacji zostają wysłane do sieci.

Powodem, dla którego przesyłane dane są kompresowane przed ich zaszyfrowaniem, jest fakt, że dane wcześniej zaszyfrowane nie kompresują się zbyt dobrze. Istota procesu kompresji jest wynajdywanie powtórzeń i wzorców występujących w treści wiadomości. Wykonanie spakowania danych wcześniej zaszyfrowanych, mających postać losowych ciągów liczb, szybciej zwiększy rozmiar przesyłanej informacji niż go zmniejszy. W tej sytuacji wykorzystując SSL zwiększalibyśmy nie potrzebnie ilość informacji krążącej w sieci.

2.5 Weryfikacja tożsamości

Jak już wcześniej wspomniałem serwer przedstawia się za pomocą certyfikatu, skąd jednak wiadomo, że osoba udostępniająca swój klucz publiczny jest rzeczywiście tym, za kogo się podaje? Przecież to żaden problem wygenerować parę kluczy, udostępnić swój klucz publiczny, i powiedzieć, że jest się osobą „X". Aby zabezpieczyć się przed tego typu sytuacją powstały urzędy potwierdzające tożsamość właściciela klucza publicznego. Jest to tak zwana zaufana trzecia strona (trusted third party), czyli ktoś obdarzany zaufaniem przez obie strony połączenia. W praktyce może to być instytucja rządowa, firma posiadająca odpowiednią akredytację lub, w ramach jednej jednostki, wydzielona komórka organizacyjna. Urzędy te są określane wspólną nazwą instytucji certyfikujących CA (Certifying Authority).

Certyfikat jest wydany, na podstawie danych przedstawionych w formularzu zgłoszeniowym. Po dokładnym sprawdzeniu tożsamości na podstawie dowodów przedstawionych przez firmę, urząd CA generuje (odpłatnie) specjalny certyfikat, podpisany swoim własnym kluczem prywatnym. Dowody przedstawiane przez firmę urzędowi zależą oczywiście od kontekstu i w przypadku sklepu internetowego, oferującego SSL będzie to dokument poświadczający istnienie danej firmy (wypis z ewidencji gospodarczej), prawo do domeny sklepu (uzyskany od firmy rejestrującej domeny, np. NASK lub Network Solutions) itp. Na przykład w systemie PGP użytkownicy sobie nawzajem certyfikują klucze publiczne.

Jeśli na skutek niedopatrzenia obowiązków przez urząd certyfikujący, ktoś poniesie szkody to owe centrum ponosi pełną odpowiedzialność finansową i prawną z tego wynikającą. Na świecie jest kilka firm upoważnionych przez RSA Data Security Inc. do wydawania certyfikatów (np. AT&T, Versign, Thawte), a ich certyfikaty są już zazwyczaj wbudowane w najpopularniejsze oprogramowanie korzystające z SSL. Aplikacja po otrzymaniu od serwera jego certyfikatu sprawdza, jaki urząd go podpisał. W tym celu szuka ona w swojej wewnętrznej bazie certyfikatu tego urzędu. Jeżeli go nie znajdzie, może spróbować poszukać go w bazie zewnętrznej do tego celu wykorzystuje się powszechnie katalogi LDAP. W przypadku, gdy serwer przedstawi certyfikat, którego wystawcą jest nieznany CA, program ostrzeże nas o tym fakcie i zapyta, czy mimo to chcemy kontynuować połączenie. Rozwiązaniem jest oczywiście zaimportowanie do certyfikatu danego CA.

Rozdział 3

Generowanie kluczy

Zanim zaczniemy programować potrzebna jest wiedza jak wygenerować certyfikat. Jest on potrzebny serwerowi, aby ten mógł przedstawić się drugiej stronie (klient nie musi tego robić).

3.1 Relacje pomiędzy kluczami

Certyfikaty są powiązane z szyfrowaniem kluczem publicznym przez zawieranie klucza publicznego. Aby wszystko mogło działać, musi istnieć odpowiedni klucz prywatny. W OpenSSL, klucze publiczne są w prosty sposób generowane z kluczy prywatnych, więc przed tym jak zostanie utworzony certyfikat lub prośba o certyfikat (Certificate Signing Request), należy najpierw stworzyć klucz prywatny. Posiadając klucz prywatny możemy z niego wygenerować certyfikat na kilka sposobów.

1. Pierwszy z nich polega na wygenerowaniu klucza publicznego dokładając do niego informację o serwerze otrzymamy Certificate Signing Request w skrócie CSR, czyli prośba o certyfikat dla serwera. W dalszej części tego rozdziału CSR będzie oznaczać prośbę o wydanie certyfikatu. Teraz możemy zgłosić nasz CSR do podpisania przez instytucję certyfikującą.

2. Druga metoda polega również na wygenerowaniu Certificate Signing Request (CSR), ale w tym przypadku instytucją certyfikującą jesteśmy sami sobie. W tej sytuacji wcześniej musimy wygenerować swój własny certyfikat CA (ang. Certifying Authority), którym będziemy podpisywać certyfikaty naszych serwerów.

3. Ostatnia metoda nie generuje CSR, ale bezpośrednio z klucza prywatnego certyfikat serwera (self-signed certyficate).

3.2 Generowanie klucza prywatnego

Klucze są podstawą szyfrowania za pomocą kluczy publicznych i PKI [1] (Public Key Infrastructure). Klucze zwykle są połączone w pary, jedna połowa będąca kluczem publicznym zaś druga połowa to klucz prywatny. Klucz prywatny zawiera informacje o kluczu publicznym i odwrotnie, więc publiczny klucz nie musi być generowany oddzielnie.

Klucze publiczne używają różnych kryptograficznych algorytmów. Najbardziej popularnymi są te skojarzone z certyfikatami RSA i DSA, w tym rozdziale zostanie przedstawione jak wygenerować każdy nich.

3.2.1 Generowanie klucza prywatnego RSA

Klucz RSA może być używany do deszyfrowania wiadomości oraz do składania podpisu cyfrowego. Wygenerowanie klucza dla algorytmu RSA jest dosyć łatwe, wszystko, co trzeba zrobić to wpisać:

```
openssl genrsa -des3 -out privkey.pem 2048
```

W poleceniu tym:

genrsa - służy do generowania klucza RSA,

-out - określa nazwę pliku, w którym znajdzie się wygenerowany CSR

-des3 - z tym wariantem, zostaniemy poproszeni o hasło chroniące. Jeśli nie chcesz, aby twój kluczowy był chroniony przez hasło, usuń flagę ' -des3' z wyżej podanego polecenia,

2048 jest wielkością klucza, w bitach. Obecnie, liczba 2048 lub wyższa jest polecana dla kluczy RSA. Mniejsza liczba bitów nie gwarantuje dostatecznego bezpieczeństwa (być może jeszcze nie teraz, ale w wkrótce klucze o mniejszej liczbie bitów nie będą bezpieczne),

Uwaga: jeśli klucz ma być użyty razem z certyfikatem dla serwera, to może się okazać, że lepiej będzie utworzyć klucz bez hasła chroniącego, w przeciwnym razie ktoś będzie musiał podawać hasło za każdym razem gdy serwer będzie chciał uzyskać dostęp do klucza. Klucz jest zabezpieczony hasłem (-des3), więc bez jego podania nie uruchomimy serwera. Aby zdjąć hasło należy wydać polecenie:

```
openssl rsa -in privatekey.pem -out privatekey.pem
```

[1] PKI - Public Key Infrstructure czyli infrastruktura klucza publicznego. Jest to skomplikowany system, pozwalający na hierarchiczne potwierdzanie tożsamości stron komunikacji (klientów i serwerów). Protokół, jest oparty o mechanizmy kryptograficzne z odpowiednimi procedurami organizacyjnymi, takimi jak certyfikacja uczestników systemu.

3.2.2 Generowanie klucza prywatnego DSA

Klucz DSA może być używany tylko i wyłącznie do generowania podpisu cyfrowego. Aby wygenerować klucz DSA potrzebne są dwa kroki. Po pierwsze, trzeba wygenerować parametry, z których zostanie utworzony klucz za pomocą polecenia:

```
openssl dsaparam -out dsaparam.pem 2048
```

Liczba 2048 jest wielkością klucza, w bitach. I ma takie same znaczenie jak w przypadku klucza RSA. Kolejnym krokiem jest utworzenie kluczy używając wcześniej przygotowanych parametrów (poszczególne klucze mogą być generowane na podstawie tych samych parametrów):

```
openssl gendsa -des3 -out privkey.pem dsaparam.pem
```

Polecenie powyżej tworzy klucz *privatekey.pem* chroniony hasłem. Jeśli klucz ma być wykorzystywany razem z certyfikatem dla serwera, to tak jak w przypadku klucza RSA lepiej uniknąć hasła chroniącego. W tym celu należy usunąć flagę „ -des3” z polecenia powyżej.

3.3 Generowanie certyfikatu.

We wszystkich przypadkach opisanych poniżej, potrzebny będzie plik konfiguracyjny oraz klucz prywatny o nazwie: privkey.pem. Plik konfiguracyjny dla OpenSSL ma nazwę openssl.cnf. Żeby używać innej nazwy, należy OpenSSL skompilować z opcją -config FILE. Można także wykorzystać tymczasowy plik konfiguracyjny, co także zostanie opisane.

3.3.1 Tymczasowy plik konfiguracyjny

Opcje konfiguracji są wyszczególniane w sekcji `[req]` pliku konfiguracyjnego. Jeśli w całym pliku konfiguracyjnym nie ma wartości w określonej sekcji wtedy przeszukiwana jest sekcja domyślna lub sekcja bez nazwy. Opcje dostępne są opisane szczegółowo poniżej.

input_password output_password
: Hasła dla wejściowego klucza prywatnego (jeśli jest wykorzystywany) i wyjściowego klucza prywatnego (gdy jest tworzony). Opcje w linii poleceń passin i passout zmieniają wartości z pliku konfiguracyjnego.

default_bits
: Wyszczególnia domyślny rozmiar klucza w bitach. Jeśli nie zostanie podany to przyjmowany jest rozmiar 512 bitów. Jest używany razem z opcją -new.

default_keyfile
Domyślna nazwa pliku do zapisania klucza prywatnego. Jeśli nie jest wyszczególniony to klucz jest wypisany na standardowe wyjście. Może być nadpisany przez opcję -keyout.

RANDFILE
Wyszczególnia nazwę zbioru, w którym jest umieszczany i czytany generator liczb losowych. Jest używany do generowania klucza prywatnego.

encrypt_key
Gdy jest ustawione na `no`, to jest generowany nieutajniony klucz prywatny. Opcja jest równoważna do przełącznika `-node` w linii poleceń.

default_md
Opcja ta określa algorytm skrótu, jaki należy użyć. Możliwe wartości to: `md5 sha1 mdc2`. Jeśli nie jest podany wtedy jest używana funkcja MD5.

string_mask
Ta opcja maskuje kodowanie pewnych znaków w pewnych polach.

req_extensions
to wyszczególnia sekcję zawierającą listę rozszerzeń do dodania do prośby o certyfikat. Wartość ta może zostać przykryta przez -reqexts w linii poleceń.

x509_extensions
Wyszczególnia sekcję zawierającą listę rozszerzeń do dodania do generowanego certyfikatu kiedy przełącznik `-x509` jest użyty. Wartość ta może zostać nadpisana przez `-extensions` w linii poleceń.

prompt
Jeśli ustawiona jest wartość `no` wówczas jest wyłączone proszenie o wypełnienie pól certyfikatu i wartości są pobierane z pliku konfiguracyjnego. Zmienia też spodziewany format sekcji: `distinguished\_name` i `attributes`.

utf8
Jeśli ustawiona jest wartość `yes` wtedy wartości w polach są interpretowane jako ciągi znaków zakodowane w UTF8, domyślne kodowanie

odbywa się w ASCII. To znaczy, że wartości w polach, czy wpisywane z konsoli, muszą być zgodne z kodowaniem UTF8.

attributes
Wyszczególnia sekcję zawierającą jakiekolwiek atrybuty prośby: jego format jest taki sam jak distinguished_name. Aktualnie jest to ignorowane.

distinguished_name
Wyszczególnia sekcję zawierającą wyszczególnione nazwy pól do szybkiego generowania świadectwa lub prośby o świadectwo (Certificate Signing Request).

Przykładowy plik konfiguracyjny może mieć postać:

```
[ req ]
 default_bits           = 1024
 default_keyfile        = privkey.pem
 distinguished_name     = req_distinguished_name
 attributes             = req_attributes
 x509_extensions        = v3_ca
 dirstring_type = nobmp
 [ req_distinguished_name ]
 countryName                    = Country Name (2 letter code)
 countryName_default            = AU
 countryName_min                = 2
 countryName_max                = 2
 localityName                   = Locality Name (eg, city)
 organizationalUnitName         = Organizational Unit Name (eg, section)
 commonName                     = Common Name (eg, YOUR name)
 commonName_max                 = 64
 emailAddress                   = Email Address
 emailAddress_max               = 40
 [ req_attributes ]
 challengePassword              = A challenge password
 challengePassword_min          = 4
 challengePassword_max          = 20
 [ v3_ca ]
 subjectKeyIdentifier=hash
 authorityKeyIdentifier=keyid:always,issuer:always
 basicConstraints = CA:true
```

Przykładowy plik konfiguracyjny zawierający wartości dla wszystkich pól:

```
RANDFILE               = \$ENV::HOME/.rnd
 [ req ]
 default_bits           = 1024
 default_keyfile        = keyfile.pem
 distinguished_name     = req_distinguished_name
 attributes             = req_attributes
 prompt                 = no
 output_password        = mypass
 [ req_distinguished_name ]
 C                      = GB
 ST                     = Test State or Province
 L                      = Test Locality
 O                      = Organization Name
 OU                     = Organizational Unit Name
 CN                     = Common Name
 emailAddress           = test@email.address
 [ req_attributes ]
 challengePassword              = A challenge password
```

3.3.2 Tworzenie Certificate Signing Request (CSR)

Aby stworzyć Certyfikat, należy rozpocząć od prośby o certyfikat. W rzeczywistości tworzymy klucz publiczny z klucza prywatnego, który dodatkowo zawiera informacje o serwerze. To wszystko stanowi Certificate Signing Request (CSR), czyli po prostu niepodpisany jeszcze certyfikat X.509. CSR jest tworzony w ten sposób:

```
openssl req -new -config path -key privkey.pem -out cert.csr
```

W poleceniu tym:

req - zadanie generowania klucza publicznego,

-config - ścieżka do pliku konfiguracyjnego (openSSL.cnf),

-key - nazwa wcześniej wygenerowanego klucza prywatnego,

-out - określa nazwę pliku, w którym znajdzie się wygenerowany CSR.

3.3.3 Tworzenie self-signed Certyficate

Tworzenie self-signed Certyfikatu jest podobne do tworzenia CSR, ale od razu jest tworzony certyfikat (ta metoda nie jest polecana do tworzenia certyfikatu CA):

```
openssl req -new -x509 -config path -key privkey.pem -out cacert.pem
-days 1095
```

Gdzie:

-x509 - zadanie generowania certyfikatu,

-config - ścieżka do pliku konfiguracyjnego (openSSL.cnf),

-key - nazwa wcześniej wygenerowanego klucza prywatnego, którego ma być wygenerowany certyfikat,

-out - określa nazwę pliku, w którym znajdzie się wygenerowany certyfikat dla naszego serwera.

3.3.4 Tworzenie certyfikatu CA (Certificate Authority)

Certyfikat CA zawiera zbiór informacji reprezentujących tożsamość danej instytucji certyfikującej. Obecność podpisu danego CA na certyfikacie serwera oznacza, że CA zaakceptował dowody przedstawione przez firmę występującą o podpis i swoim certyfikatem poświadcza autentyczność serwera.

Aby wygenerować własny certyfikat CA należy najpierw wygenerować klucz prywatny poleceniem:

```
openssl rsa -in privatekey.pem -out privatekey.pem
```

Następnie generujemy CSR podając dane dotyczące naszego urzędu certyfikującego przez wydanie polecenia:

```
openssl req -new -config path -key privkey.pem -out cacert.pem
```

Teraz bierzemy podpisywany CSR, podpisujący klucz prywatny i wykonujemy polecenie:

```
openssl x509 -req -signkey cakey.pem -in cacert.csr -out cacert.pem
-days 1095
```

-signkey - nazwa wcześniej wygenerowanego klucza prywatnego, którego ma być wygenerowany certyfikat,

-out - określa nazwę pliku, w którym znajdzie się wygenerowany certyfikat dla naszego serwera,

days 1095 - liczba dni, przez które będzie ważny certyfikat.

W tej chwili posiadamy już certyfikat o nazwie cacert.pem i możemy nim podpisać CSR naszego serwera otrzymując w rezultacie certyfikat. W tym celu należy:

```
openssl x509 -req -CA cacert.pem -Cakey cakey.pem  -CAcreateserial
-in cert.csr -out cert.pem
```

Gdzie:

-x509 - zadanie generowania certyfikatu,

-config - ścieżka do pliku konfiguracyjnego,

-key - nazwa wcześniej wygenerowanego klucza prywatnego, którego ma być wygenerowany certyfikat,

-out - określa nazwę pliku, w którym znajdzie się wygenerowany certyfikat dla naszego serwera,

days 1095 - liczba dni, przez które będzie ważny certyfikat.

Rozdział 4

Teapop + SSL

OpenSSL jest rozpowszechnioną na otwartej licencji biblioteką kryptograficzną. Zapewnia ona także obsługę bezpiecznych protokołów sieciowych Secure Socket Layer v2/v3 (SSLv2/SSLv3) i Transport Layer Security v1 (TLSv1). Dostarcza, więc odpowiednich narzędzi, aby wbudować obsługę protokołu SSL do Teapop-a.

W skład OpenSSL zawiera liczne *pliki nagłówkowe*, między innymi: ssl.h, crypto.h, err.h, rsa.h, x509.h, pem.h. Każdy z tych plików dostarcza odrębne funkcje przeznaczone do różnych zadań. Opis, jakiego typu są to funkcje znajduje się poniżej.

ssl

Dostarcza bogaty zbiór funkcji API dla protokołów Secure Sockets Layer (SSL v2/v3) i Transport Layer Security (TLS v1). Obecnie biblioteka ssl liczy sobie 214 funkcji API.

crypto

Dostarcza szeroki wachlarz algorytmów kryptograficznych używanych w różnych standardach Internetu. Usługi udostępniane przez tą bibliotekę są użyte przez OpenSSL w implementacji SSL, TLS, SSH i OpenPGP.

err

Kiedy wywołanie funkcji z biblioteki OpenSSL zawiedzie, zwykle jest to sygnalizowane przez zwróconą wartość i ustawianie kodu błędu na stosie błędów. Biblioteka `err` dostarcza funkcje do uzyskania czytelnych komunikatów o błędzie. Kod błędu zawieraja informację gdzie wystąpił błąd, i co poszło nie tak.

rsa

Jak sama nazwa wskazuje biblioteka dostarcza funkcji, związanych z szyfrowaniem z kluczem publicznym RSA.

x509

Certyfikat X.509 jest strukturą zawierającą informacji o osobie, urządzeniu , lub czymkolwiek, co można sobie wyobrazić.

4.1 Opis funkcji z pliku `pop_ssl.c`

Dobrym zwyczajem jest podzielenie kodu na części odpowiedzialne za specyficzne zadania. Stosując się do tej reguły programistycznej, funkcje i zmienne odpowiedzialne za współpracę serwera Teapop z biblioteką OpenSSL, umieściłem w osobnym pliku: `pop_ssl.c`. Jest to rozwiązanie nie tylko eleganckie, ale także ułatwiające znalezienie funkcji realizującej jakieś określone zadanie. W działaniu protokołu SSL można wyróżnić kilka etapów:

1. Na początku należy zaimplementować protokoły Secure Sockets Layer (SSL v2/v3) i Transport Layer Security (TLS v1). Jest to realizowane przez funkcje API.

2. Następnie tworzony jest obiekt `SSL_CTX` (przy pomocy funkcji `SSL_CTX_new()`) będący fundamentem do stworzenia połączenia TLS/SSL. W obiekcie tym ustawiane są opcje połączenia, wczytywany jest certyfikat i klucz prywatny, ustawiana jest lista obsługiwanych algorytmów itp.

3. Kolejnym etapem jest stworzenie struktury `SSL` (przy pomocy funkcji `SSL_new()`) na podstawie obiektu `SSL_CTX` i dowiązanie do niej połączenia TCP.

4. Teraz wykorzystując `SSL_accept()` dochodzi do uzgadniania połączenia TLS/SSL (ang. handshake). Gdy proces się powiedzie można wysyłać i odbierać dane odpowiednio za pomocą `SSL_write()` i `SSL_read()`.

5. Na końcu połączenie TLS/SSL jest zamykane i zwalniana jest pamięć zajmowana przez struktury `SSL` i `SSL_CTX`.

Poniżej przedstawiam plik `pop_ssl.h`, w którym zdefiniowane są stałe, zmienne oraz umieszczone są prototypy funkcji z pliku `pop_ssl.c`.

```
/* $Id: //depot/Teapop/0.3/teapop/teapop.c#8 $ */

/*
 * Copyright (c) 1999-2003 ToonTown Consulting
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 * 3. Neither the name of the company nor the names of its contributors
```

```
 *    may be used to endorse or promote products derived from this software
 *    without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR ``AS IS'' AND ANY EXPRESS OR
 * IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
 * OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED.
 * IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT,
 * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
 * NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
 * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
 * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF
 * THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

#include <syslog.h>

#include <openssl/rsa.h>
#include <openssl/crypto.h>
#include <openssl/x509.h>
#include <openssl/pem.h>
#include <openssl/ssl.h>
#include <openssl/err.h>

#ifndef CERTF
#define CERTF ETC_DIR"/popcert.pem"
#endif

#ifndef KEYF
#define KEYF ETC_DIR"/popkey.pem"
#endif

extern SSL* ssl;
extern SSL_CTX* ctx;

int ssl_init ();
int ssl_connect (int socket);
void ssl_end();
void ssl_print_err();
```

Aby zacząć korzystać z funkcji oferowanych przez OpenSSL, trzeba dołączyć pliki nagłówkowe. Odbywa się to za pomocą dyrektywy `#include`. Oprócz plików nagłówkowych OpenSSL, importowany jest także plik `syslog.h`. Zawiera on niezbędne funkcje do wypisywania komunikatów o błędach, zdarzeniach do systemowego pliku z logami. Należy również zdefiniować kilka stałych. Biblioteka OpenSSL musi mieć dostęp do pliku z certyfikatem oraz kluczem. Będę się do nich odwoływać przez stałe `CERTF` oraz `KEYF`. Chociaż są to stałe, to użytkownik będzie mógł ustawić ich wartość podczas konfiguracji serwera Teapop. Jeśli tego nie zrobi zostaną im przypisane wartości domyślne (`ETC_DIR"/popcert.pem"` i `ETC_DIR"/popkey.pem"`). Odbywa się to za pomocą kompilacji warunkowej. Jeśli certyfikat (warunek `#ifndef CERTF`) lub klucz (warunek `#ifndef KEYF`) nie zostały zdefiniowane podczas konfiguracji oprogramowania Teapop, to dla stałych przypisywana jest wartość domyślna. Następnym krokiem jest deklaracja zmiennych, które będą używane w bibliotece OpenSSL. Deklarowana jest zmienna ctx typu `SSL_CTX` oraz zmienna `ssl` typu `SSL`. Struktura `SSL_CTX`, jest tworzona raz podczas działania programu, natomiast obiekt `SSL` jest tworzony przez serwer podczas każdego połącze-

nia i zawiera m.in. wartości wynegocjowane podczas uzgadniania połączenia. Zmienne `ctx` i `ssl` są zmiennymi globalnymi.

W dalszej części znajduje się opis funkcji z pliku `pop_ssl.c` wraz z kodem źródłowym.

4.1.1 ssl_print_err()

```
void ssl_print_err(){
        int ssl_err;         /* SSL error code */
        char err_str[512];  /* string representing the SSL error code*/

        /* retrive the latest SSL error code */
        ssl_err = ERR_peek_last_error();

        /* generate string representing the error code */
        ERR_error_string_n(ssl_err, err_str, 500);

        /* report to syslog facility */
        syslog(LOG_ERR, "SSL: %d: %s", ssl_err, err_str);
}
```

Opis:

Chyba nie można bagatelizować takiej sprawy jak raportowanie o błędach. Informowacje, w którym miejscu pojawił się błąd i co to był za błąd, potrafią wiele wyjaśnić. Przy niepoprawnym działaniu można zawsze odwołać się do logów i sprawdzić jaka jest tego przyczyna. Właśnie ta funkcja wypisuje komunikaty o błędach do logów Teapop-a. Działanie jej jest następujące:

1. Pobierany jest ostatni kod błędu.
2. Numer błędu przekształcany jest za pomocą `ERR_error_string_n()` do zrozumiałej dla człowieka postaci.
3. Informacja o błędzie jest zapisywana w logu.

4.1.2 ssl_init()

```
int ssl_init ()
{
        SSL_METHOD *meth;

         /* register the error strings for libcrypto and libssl functions */
        SSL_load_error_strings();

        /* registers the available ciphers and digests */
        SSLeay_add_ssl_algorithms();

        /* TeaPop will understand the SSLv2, SSLv3, and TLSv1 protocols */
        meth = SSLv23_server_method();

        /* create a new SSL_CTX structure */
        ctx = SSL_CTX_new(meth);
        if (!ctx)
        {
             ssl_print_err();
```

```
            return 1;
        }

        /* This function loads the certificate from CERTF into the SSL_CTX object */
        if (SSL_CTX_use_certificate_file(ctx, CERTF, SSL_FILETYPE_PEM) <= 0)
        {
            ssl_print_err();
            syslog(LOG_ERR, "Error loading certificate file: %s", CERTF);
            return 2;
        }

        /* This function load the private key from KEYF into ctx */
        if (SSL_CTX_use_PrivateKey_file(ctx, KEYF, SSL_FILETYPE_PEM) <= 0)
        {
            ssl_print_err();
            syslog(LOG_ERR, "Error loading private key file: %s", KEYF);
            return 3;
        }

        /* Check the consistency of a private key with the certificate */
        if (!SSL_CTX_check_private_key(ctx))
        {
            ssl_print_err();
            return 4;
        }

        return 0;
}
```

Opis:

Funkcja ta inicjuje bibliotekę OpenSSL. Na początku ładowane są kody błędów (`SSL_load_error_strings()`) oraz inicjowana jest biblioteka kryptograficzna (`SSLeay_add_ssl_algorithms()`). Następnie ustalany jest protokołu, jaki będzie obsługiwany przez serwer Teapop. Odbywa się to przez wywołanie funkcji `SSLv23_server_method()`. Użyta funkcja zapewnia komunikację za pomocą protokołów SSL v1/v2 i TLSv1. Następnie tworzona jest zmienna `ctx` reprezentująca strukturę `SSL_CTX`. Do zmiennej tej wczytywany jest certyfikat oraz klucz publiczny i sprawdzana jest spójność klucza z certyfikatem. Jeśli w wyniku wywołania którejkolwiek z funkcji biblioteki OpenSSL wystąpi błąd, jest to odnotowywane w logu przy pomocy `ssl_print_err()`.

Zwracane wartości:

0
: Zainicjowanie biblioteki OpenSSL przebiegło pomyślnie.

>0
: Wystąpił błąd, w logu znajduje się szczegółowy komunikat. informacji

4.1.3 ssl_connect()

```
int ssl_connect (int socket)
{
        int ssl_err;        /* SSL error code */
```

```
        /* create a new SSL structure */
        ssl = SSL_new(ctx);
        if (ssl == NULL) {
             syslog(LOG_ERR,"SSL_new(): can't create SSL structure");
             return 5;
        }

        /* connect the SSL object with a socket file descriptor */
        SSL_set_fd(ssl, socket);

        /* wait for client to initiate a TLS/SSL handshake */
        ssl_err = SSL_accept(ssl);
        if (ssl_err == -1) {
              ssl_print_err();
              return 6;
        }
        return 0;
}
```

Opis:

Funkcja tworzy zmienną `ssl`, reprezentującą połączenie SSL. Jest do tego wykorzystana funkcja `SSL_new()`. Jednak to nie wystarczy żeby komunikacja SSL była możliwa. Do struktury `SSL` należy jeszcze dołączyć deskryptor połączenia TCP. W tym celu wołana jest funkcja `SSL_set_fd()`. Jej pierwszym argumentem jest obiekt `SSL`, drugim deskryptor połączenia TCP. Samo połączenie nawiązywane jest poza funkcją `ssl_connect()`, a jego identyfikator przekazany jest do funkcji jako parametr o nazwie `socket`. Pozostało jeszcze uzgodnienie połączenia SSL z klientem, co realizuje funkcja `SSL_accept()`. Po jej wywołaniu Teapop będzie oczekiwał na klienta.

4.1.4 ssl_end()

```
void ssl_end(){
        /* remove the SSL structure */
        SSL_free(ssl);

        /* remove the SSL_CTX object pointed to by ctx */
        SSL_CTX_free(ctx);
}
```

Opis:

Funkcja ta zwalnia pamięć zajmowaną przez zmienne globalne `ssl` i `ctx`. Odpowiedzialne są za to funkcje `SSL_free(ssl)` i `SSL_CTX_free(ctx)`. Bardzo ważne jest, aby nie zaśmiecać pamięci i usuwać zmienne, które nie będą już wykorzystywane przez program. Jest to konieczne, ponieważ w końcu doszłoby do wyczerpania zasobów pamięci i przeciążenia systemu operacyjnego, pod którym uruchomiony jest Teapop.

4.2 Zmiany w kodzie Teapop-a

Zanim zacząłem programować potrzebna mi była wiedza na temat jak wygenerować certyfikat dla serwera Teapop. Jest on potrzebny serwerowi, aby mógł się przedstawić dla klienta. Każdy taki certyfikat musi być podpisany przez instytucję certyfikującą. Ponieważ koszt takiej operacji w przypadku znanych firm certyfikujących jest dość znaczny, rozwiązanie takie było dla mnie nie przydatne. Na szczęście OpenSSL oferuje coś takiego jak certyfikaty *self-signed*, czyli podpisane przez tą samą osobę, która wystawia CSR. Potrzebowałem, więc swojego własnego certyfikatu CA, którym mógłbym podpisać certyfikat Teapop-a. Gdy już posiadałem wygenerowany certyfikat, musiałem zrozumieć sposób działania serwera Teapop. Ponieważ nie znalazłem żadnej dokumentacji na stronie Teapop-a, więc nie miałem innego wyjścia jak przejrzeć kod źródłowy. To było moje pierwsze, poważne zetknięcie się z językiem C. Zatem co rusz sięgałem do stron podręcznika systemu Solaris9[1], aby sprawdzić, do czego służą poszczególne funkcje.

Po analizie kodu serwera Teapop stwierdziłem, że działa on w dwóch trybach:

1. Tryb standalone.
2. Jako usługa procesu inetd.

Aby uruchomić Teapop-a w trybie standalone należy użyć opcji `-s`, natomiast aby uruchomić go przez program inetd w pliku `/etc/inetd.conf` należy dodać następującą linijkę:

```
110 stream tcp nowait root /usr/local/libexec/teapop teapop
```

gdzie `/usr/local/libexec/teapop` jest ścieżką do programu Teapop.

W zależności od sposobu uruchomienia Teapop-a sposób działania serwera nieznacznie się różni. Sprowadza się to do wywołania różnych funkcji w zależności od trybu pracy. Jeśli Teapop zostanie uruchomiony przez inetd to jest wywoływana funkcja `teapop()`, jeśli zaś zostanie uruchomiony w trybie standalone to wołana jest funkcja `teapop_s()`. Oto odpowiedni fragment kodu:

```
if (standalone == 1)
        retval = teapop_s(&pinfo);
else
        retval = teapop(&pinfo);
```

Jak widać na powyższym listingu za pomocą instrukcji warunkowej `if` w zależności od sposobu uruchomienia podejmowana jest decyzja, którą z funkcji trzeba uruchomić. Choć są tutaj uruchamiane dwie różne funkcje, to ostatecznie wszystko sprowadza się do wywołania funkcji `teapop()`, bo w `teapop_s()` jest odwołanie do `teapop()`. Więc powstaje pytanie, po co to całe zamieszanie? Dzieje się tak, dlatego, że gdy Teapop jest uruchamiany, przez inetd to

[1]System operacyjny pod którym testowałem zmodyfikowaną wersję serwera Teapop

otrzymuje już otwarte połączenie TCP, natomiast, gdy jest uruchomiony w trybie standalone sam musi uzgodnić połączenie TCP z klientem. Dopiero gdy będzie miał otwarte połączenie i ustalony identyfikator połączenia, wywołuje funkcję `teapop()`. Tyle zamieszania o jakieś połączenie, ale nie napisałem jeszcze do czego jest mi ono potrzebne. Otóż warstwa SSL znajduje się pomiędzy warstwą aplikacji a warstwą TCP (rysunek 2.2) i żeby warstwy mogły ze sobą współpracować za pomocą protokołu SSL, konieczne jest wcześniejsze ustanowienie połączenia TCP. Dlatego musiałem, odszukać w kodzie Teapopa deskryptor połączenia TCP aby dowiązać go do struktury SSL. Tak, więc ostatecznie nie zależnie od sposobu uruchomienia wszystko sprowadza się do wywołania funkcji `teapop()`. Do funkcji tej przekazywana jest zmienna `pinfo` typu POP_INFO, która zawiera identyfikator połączenia. Deskryptor ten jest pamiętany w polu `insck` zmiennej `pinfo` i to właśnie ten deskryptor należy dowiązać do zmiennej SSL. Skoro mam już ustalony identyfikator połączenia mogę przystąpić do zainicjowania biblioteki OpenSSL.

W pliku `teapop.c` należy zainicjować bibliotekę OpenSSL, co jest realizowane przez funkcję `ssl_init()`. Jednak żeby móc skorzystać z tej funkcji najpierw trzeba dołączyć plik `pop_ssl.h`. Odbywa się to w następujący sposób:.

```
#ifdef WITH_SSL
    #include "pop_ssl.h"
#endif
```

Instrukcja `#ifdef WITH_SSL` to makropolecenie preprocesora, które dołączy plik `pop_ssl.h` tylko wtedy, gdy Teapop ma obsługiwać SSL (wartość `WITH_SSL` jest definiowana podczas konfiguracji). Po dołączeniu tego pliku można korzystać ze zmiennych globalnych `ssl` i `ctx` oraz wszystkich funkcji zdefiniowanych w pliku `pop_ssl.c`. Można, zatem wywołać funkcję `ssl_init()`. Wywołanie tej funkcji powinno nastąpić przed funkcją `teapop()`, a tym samym, również przed funkcją `teapop_s()`. Miejsce wywołania funkcji `ssl_init()` jest zaprezentowane na listingu:

```
#ifdef WITH_SSL
        /* initialize SSL global CTX structure */
         if (ssl_init() != 0){
                 exit(1);
         }
#endif

         if (standalone == 1)
                 retval = teapop_s(&pinfo);
         else
                 retval = teapop(&pinfo);
```

Oczywistym jest, że bibliotekę OpenSSL należy zainicjować tylko wtedy gdy Teapop ma obsługiwać protokół SSL, stąd kompilacja warunkowa.

Mam już zaimplementowaną bibliotekę OpenSSL oraz utworzony obiekt `SSL_CTX`. Teraz należy utworzyć obiekt `SSL` i dowiązać do niego identyfikator połączenia. Zrobi to funkcja `ssl_connect()`, do której jako parametr wystar-

czy przekazać wartość `pinfo->insck`. Tak, więc wywołanie funkcji będzie miało następującą postać `ssl_connect(pinfo->insck)`. Powstaje pytanie gdzie tą linijkę należy umieścić. Na pewno w funkcji `teapop()`, ponieważ do niej ostatecznie sprowadza się działanie programu i to tam jest dostępny deskryptor połączenia. Dodatkowo wywołanie powinno nastąpić jeszcze przed wymianą danych między serwerem a klientem, a więc także przed autoryzacją. Jak zostało to rozwiązane prezentuje poniższy listing:

```
/* Make sure stdin is a socket */
 if ((retval = pop_socket_init(pinfo)) != 0) {
         if (retval == 2) {
                 fprintf(stderr,"connection refused by tcp_wrappers\r\n");
         } else {
                 fprintf(stderr, "stdin is not a socket\r\n");
                 fprintf(stderr, "If you want to run teapop in standalone " \
                     "mode you have to use the -s argument.\r\n");
         }
         return (retval);
 }

#ifdef WITH_SSL
 /* initialize SSL structure per connection */
 if (ssl_connect(pinfo->insck) != 0){
         exit(1);
 }
#endif

switch (pop_auth(pinfo)) {
 case 0:         /* USER IS AUTH'ED */
         if ((retval = pop_lock_maildrop(pinfo, 1)) != 0) {
                 if (retval == 3) {
```

Przed wywołaniem funkcji `ssl_connect()` znajduje się blok odpowiedzialny za sprawdzenie poprawności połączenia. Sprawdzenie to wykonuje funkcja `pop_socket_init()`. Gdy funkcja zwróci wartość 0 oznacza to, że wszystko jest w porządku i można dowiązać identyfikator połączenia do zmiennej `ssl`. Jeszcze należy zadbać o to by Teapop z SSL, działał na odpowiednim porcie. Przyjętym standardem jest, aby protokół POP3 z obsługą SSL wykorzystywał port TCP o numerze 995. W tym celu w pliku `teapop.h` została zdefiniowana stała `SPOP3PORT` :

```
#define POP3PORT        110     /* Port to listen to in standalone mode */
#define SPOP3PORT       995     /* Port to listen to in standalone mode
                                   with SSL enabled*/
```

Stała `SPOP3PORT` określa port, na którym będzie odbywała się komunikacja podczas szyfrowanej sesji POP3. Definicja ta nie przypadkowo znalazła się obok definicji stałej `POP3PORT`. To właśnie `POP3PORT` określa numer portu, na którym Teapop standardowo przyjmuje połączenia w trybie standalone. Również stała `SPOP3PORT` będzie wykorzystywana tylko w trybie standalone, ale gdy Teapop będzie obsługiwał SSL. Dzieje się tak, bo to właśnie w tym trybie, w funkcji `teapop_s()` jest ustalane połączenie TCP. Odpowiedzialna jest za to funkcja `pop_socket_bind()`, której definicja znajduje się w pliku `pop_socket.c`. Tam też trzeba wprowadzić zmiany. W funkcji `pop_socket_bind()`, zdefiniowana jest zmienna `port` określająca port, na

którym Teapop oczekuje na połączenia. Wykorzystując ponownie kompilację warunkową, w zależności od rodzaju konfiguracji Teapop-a do zmiennej port należy przypisać wartość stałej SPOP3PORT lub POP3PORT.

```
#ifdef WITH_SSL
        if (port == 0) port = SPOP3PORT;
#else
        if (port == 0) port = POP3PORT;
#endif
```

Teraz, gdy Teapop odbiera połączenia na odpowiednim porcie można zająć się pozostałymi zmianami. Zmiany te będą dotyczyły funkcji odpowiedzialnych za wysyłanie i odbieranie danych, które są zdefiniowane w pliku `pop_socket.c`. Funkcje, służące do wysyłania danych to `pop_socket_send()` i `pop_socket_rawsend()`, zaś funkcja `pop_wait_for_commands()` służy do odbierania danych . Przed wysyłaniem danych do klienta, Teapop powinien przekazać je do warstwy SSL, która po zaszyfrowaniu danych, przekaże je do warstwy transportowej. Z kolei, gdy dane są odbierane, zanim trafią do Teapop-a, muszą przejść przez warstwę SSL, gdzie tym razem zostaną odszyfrowane. Reprezentacją warstwy SSL jest zmienna `ssl`, więc do `pop_socket.c` należy dołączyć plik `pop_ssl.h`.

Zmodyfikowane funkcje do wysyłania danych zostały przedstawione na poniższym listingu:

```
void
#if __STDC__
pop_socket_send(FILE * fd, const char *fmt, ...)
#else
pop_socket_send(fd, fmt, va_alist)
FILE *fd;
char *fmt;
va_dcl
#endif
{
        char buf[512];

        va_list ap;
#if __STDC__
        va_start(ap, fmt);
#else
        va_start(ap);
#endif
        vsnprintf(buf, 508, fmt, ap);
        if (!(buf[strlen(buf - 2)] == '\r' && buf[strlen(buf - 1)] == '\n'))
                strcat(buf, "\r\n");
        va_end(ap);

#ifdef WITH_SSL
        SSL_write(ssl, buf, strlen(buf));
#else
        (void) fputs(buf, fd);
        (void) fflush(fd);
#endif
}

int
pop_socket_rawsend(fd, buffer)
FILE *fd;
char *buffer;
{
```

```
        register int retval;

#ifdef WITH_SSL
        retval = SSL_write(ssl, buffer, strlen(buffer));
#else
        retval = fputs(buffer, fd);
#endif
        return (retval <= -1 ? 1 : 0);
}
```

Nie będę szczegółowo omawiał działania tych funkcji, ponieważ nie jest to potrzebne do zrozumienia wprowadzonych zmian. Najważniejsza jest tutaj funkcja `fputs()`, bo to ona wysyła dane do klienta. Dlatego jeśli SSL jest włączone to, zamiast `fputs()`, powinna zostać użyta funkcja `SSL_write()`(patrz dodatek B.1.16), która wysyła dane po uprzednim zaszyfrowaniu. Funkcja ta używa zmiennej globalnej `ssl`, w której pamiętane są parametry połączenia z klientem. Do zmiennej `ssl` został już dowiązany identyfikator połączenia, zatem zmienna `fd` oznaczająca deskryptor połączenia, jest nie potrzebna, jeśli Teapop obsługuje SSL.

Należy również przekształcić funkcję `pop_wait_for_commands()`, tak aby odbierane dane były odszyfrowywane. Zmieniona funkcja do odbierania danych została przedstawiona poniżej:

```
int
#if __STDC__
pop_wait_for_commands(int timeout, int size, char *arguments, ...)
#else
pop_wait_for_commands(timeout, size, arguments, va_alist)
char *arguments;
int timeout, size;
va_dcl
#endif
{
        char buf[300], slask[512];
        static char *ptr;
        int arg;
        static va_list ap;
#ifdef WITH_SSL
        int sslread;
#endif
#if __STDC__
        va_start(ap, arguments);
#else
        va_start(ap, arguments);
#endif
        if ((ptr = va_arg(ap, char *)) == NULL)
        {
                va_end(ap);
                return (-1);
        }

        for (;;) {
                alarm(timeout);
                if (setjmp(env)) {
                        va_end(ap);
                        return (0);
                }
#ifdef WITH_SSL
                if ((sslread = SSL_read(ssl, buf, 255)) <= 0) {
#else
                if ((fgets(buf, 255, stdin)) == NULL) {
```

```
#endif

                        if (feof(stdin))
                                pop_signal_sigpipe(SIGPIPE);
                        else
                                return (-1);
                }

#ifdef WITH_SSL
                /* insert termination char */
                buf[sslread] = '\0';
#endif


                if (buf[strlen(buf) - 1] != '\n') {
                        slask[0] = '\0';
                        while (slask[strlen(slask) - 1] != '\n')
#ifdef WITH_SSL
                                SSL_read(ssl, slask, 500);
#else
                                fgets(slask, 500, stdin);
#endif
                }
                alarm(0);

                while (buf[strlen(buf) - 1] == '\n' ||
                    buf[strlen(buf) - 1] == '\r') buf[strlen(buf) - 1] = '\0';
                arg = 1;
                while (ptr != NULL) {
                        if (!strncasecmp(ptr, buf, strlen(ptr)) &&
                            (strlen(buf) == strlen(ptr) ||
                                buf[strlen(ptr)] == ' ')) {
                                strncpy(arguments,
                                    buf + strlen(ptr) + 1, size - 1);
                                arguments[size - 1] = '\0';
                                return (arg);
                        }
                        ptr = va_arg(ap, char *);
                        arg++;
                }
                ptr = strchr(buf, ' ');
                if (ptr != NULL)
                        *ptr = '\0';
                fprintf(stderr, "%s " POP_UNKNOWN "\r\n", POP_ERR, buf);
                va_end(ap);
                va_start(ap, arguments);
                ptr = va_arg(ap, char *);
        }

        va_end(ap);
        return (-1);
}
```

Podobnie jak przy wysyłaniu danych również przy odbiorze trzeba jedynie zamienić funkcję odpowiedzialną za czytanie danych. Tutaj funkcją, która do tego służy jest `fgets()` i to zamiast niej trzeba użyć funkcji `SSL_read()` (patrz. dodatek B.1.15). Funkcja pobiera od obiektu `ssl`, dane otrzymane od klienta i zapisuje je do bufora `buf`. Jednak w buforze nie jest ustawiany znak końca ciągu. Dlatego została zdefiniowana zmienna `sslread`, do której jest przypisana wartość zwrócona przez `SSL_read()`, oznaczająca ilość odczytanych bajtów. Wykorzystując to ustawiamy koniec ciągu:
`buf[sslread] = '\0'`.

Na koniec po zamknięciu połączenia z klientem, należy zadbać o usunięcie obiektów `SSL_ctx` oraz `SSL`, które są reprezentowane przez zmienne `ctx` i `ssl`. W tym celu należy wywołać funkcję `ssl_end()`. Odbywa się to w funkcji `teapop()`, na sam koniec jej działania i wygląda następująco:

```
        default:        /* NEVER REACHED */
                break;
        }

#ifdef WITH_SSL
        /* free memory used by SSL & SSL_CTX objects */
        ssl_end();
#endif

        return (0);
} /* end teapop() */
```

4.3 Nowe opcje konfiguracji serwera Teapop

Posiadając sam kod źródłowy programu nie można go jeszcze uruchamiać. Aby było to możliwe najpierw należy skompilować pliki źródłowe, w rezultacie, czego otrzymamy program wynikowy, który będzie można uruchomić. W przypadku pojedynczego pliku kompilacja w cale nie jest trudna wystarczy uruchomić kompilator i podać nazwę kompilowanego pliku. Jeśli kompilacja dotyczy pliku napisanego w języku C można wydać następujące polecenie:

```
gcc plik_zrodlowy.c -o program
```

Gdzie opcja `-o` oznacza plik wynikowy kompilacji. Jeśli wszystko przebiegnie pomyślnie, to po wydaniu polecenia `./program` zostanie uruchomiony program.

Jednak w przypadku Teapop-a nie jest to już takie proste. Ponieważ na cały projekt składa się około 65 plików z kodem źródłowym. To wszystko należałoby skompilować, pamiętając dodatkowo o prawidłowej kolejności kompilowanych plików. Dlatego też przez autorów programów przygotowywane są pliki, które zawierają informację na temat konfiguracji i kompilacji. Najważniejsze z nich to `configure` i `Makefile`. Skrypt `configure`, sprawdza istnienie wymaganych składników systemowych potrzebnych do zainstalowania programu. Zaś plik `Makefile` zawiera informacje dla programu *make*, o tym w jakiej kolejności i które pliki należy skompilować. Plik `Makefile` zawiera wiele standardowych reguł, takich jak: `make all`, `make install`, `make uninstall`.

Zarówno plik `configure` jak `Makefile` są w dużej mierze generowane automatycznie na podstawie innych plików. Skrypt `configure` jest tworzony z pliku `configure.in` za pomocą programu *autoconf*. W przypadku `Makefile` sprawa jest nieco bardziej skomplikowana, a mianowicie:

1. Najpierw tworzony jest plik `Makefile.am`

2. Następnie z pliku `Makefile.am` przygotowywany jest plik `Makefile.in` za pomocą *automake*.

3. Ostatecznie skrypt `configure` tworzy na podstawie `Makefile.in` plik `Makefile`.

Więc aby umożliwić kompilację Teapop-a z obsługą SSL lub bez obsługi SSL, należy zmodyfikować pliki `configure.in` oraz `Makefile.am`. W Teapopie nie było jednak pliku `Makefile.am`, więc ograniczyłem się do zmian w pliku `Makefile.in`. Zmiany, których dokonałem w tych plikach, są przedstawione w dalszej części rozdziału.

4.3.1 Zmiany w pliku configure.in

Jak już wspomniałem, plik ten jest podstawą do wygenerowania skryptu configure, sprawdzającego w systemie obecność składników potrzebnych do zainstalowania oprogramowania. W przypadku Teapopa i SSL, powinna być zainstalowana biblioteka OpenSSL. Funkcję realizującą to zadanie zaczerpnąłem z pliku configure.in jednej z najpopularniejszych baz danych, a mianowicie z bazy MySQL. Jest to program opensource, więc można zupełnie legalnie to zrobić. Poza tym, po co pisać coś, co już zostało zrobione. Funkcją, która sprawdza obecność zainstalowanej biblioteki OpenSSL jest `FIND_OPENSSL`. Myślę, że nazwa jest dość wymowna. Cała definicja tej funkcji znajduje się w dodatku C na stronie 60.

Sprawdzenie będzie wykonane, gdy skrypt `configure` zostanie uruchomiony z opcją `--with-ssl`, w przeciwnym wypadku biblioteka OpenSSL nie będzie potrzebna. Jednak to nie wszystkie opcje, które zostały dołączone do skryptu konfiguracyjnego. Poniżej zamieszczam fragment pliku `configure.in` w którym widać, jakich opcji można użyć przy konfiguracji Teapopa z obsługą SSL:

```
dnl OpenSSL support
AC_MSG_CHECKING([for OpenSSL])
AC_ARG_WITH(openssl,
            [  --with-openssl          enable the OpenSSL support],
            openssl="$withval",
            openssl=no)

AC_ARG_WITH(openssl-includes,
            [  --with-openssl-includes=DIR
                        find OpenSSL headers in DIR],
            openssl_includes="$withval",
            openssl_includes="")

AC_ARG_WITH(openssl-libs,
            [  --with-openssl-libs=DIR
                        find OpenSSL libraries in DIR],
            openssl_libs="$withval",
            openssl_libs="")

AC_ARG_WITH(openssl-certificate,
            [  --with-openssl-certificate=FILE
```

```
                    use FILE as the certificate for SSL connections (default is PREFIX/etc/popcert.pem)],
            openssl_certificate="$withval",
            openssl_certificate="${prefix}/etc/popcert.pem")

AC_ARG_WITH(openssl-privatekey,
            [  --with-openssl-privatekey=FILE
                    use FILE as the private key for SSL connections (default is PREFIX/etc/popkey.pem)],
            openssl_privatekey="$withval",
            openssl_privatekey="${prefix}/etc/popkey.pem")
```

Po krótce wyjaśnię działanie polecenie `AC_ARG_WITH` na przykładzie

```
AC_ARG_WITH(openssl,
            [  --with-openssl          enable the OpenSSL support],
            openssl="$withval",
            openssl=no)
```

Opis poszczególnych opcji znajduje się w tabeli poniżej:

Opcja	**Opis**
`--with-openssl`	Włącza w Teapopie obsługę protokołu SSL. Jeśli nie zostanie użyta ta opcja, podawanie jakiegokolwiek z poniższych parametrów nie będzie miało sensu.
`--with-openssl-includes=DIR`	Jest to bardzo ważne, aby przy nie standardowej instalacji biblioteki OpenSSL, można było podać ścieżkę do katalogu `include`.
`--with-openssl-libs=DIR`	Ten parametr określa ścieżkę do katalogu `lib`, zawierającego niezbędne biblioteki OpenSSL do kompilacji Teapopa.
`--with-openssl-certificate=FILE`	Parametr ten określa plik z certyfikatem dla Teapop. Jeśli podczas konfiguracji parametr zostanie pominięty, certyfikat będzie szukany pod nazwą (`${prefix}/etc/popcert.pem`).
`--with-openssl-privatekey=FILE`	Opcja ta umożliwia określenie pliku z kluczem prywatnym. Jeśli podczas konfiguracji parametr zostanie pominięty, to ustalona zostanie wartość domyślna dla tego parametru (`${prefix}/etc/popkey.pem`).

Tabela 4.1: Opis opcji do konfiguracji Teapopa z obsługą SSL

Wartości zostały nadane odpowiednim parametrom, jednak wciąż nie ma gwarancji, że ścieżka do pliku z certyfikatem jest prawidłowa i czy biblioteka OpenSSL jest rzeczywiście zainstalowana. Test sprawdzający poprawność parametrów i obecność w systemie biblioteki OpenSSL jest przedstawiony poniżej:

```
if test "$openssl" = "yes"
then
  FIND_OPENSSL($openssl_includes, $openssl_libs)
```

```
  AC_MSG_RESULT(yes)
  LDFLAGS="$LDFLAGS -L$OPENSSL_LIB -R$OPENSSL_LIB"
  LIBS="$LIBS -lcrypto -lssl"
  # Don't set openssl_includes to /usr/include as this gives us a lot of
  # compiler warnings when using gcc 3.x
  if test "$OPENSSL_INCLUDE" != "-I/usr/include"
  then
      CFLAGS="$CFLAGS $OPENSSL_INCLUDE"
  fi
  if test "$OPENSSL_KERBEROS_INCLUDE"
  then
      CFLAGS="$CFLAGS -I$OPENSSL_KERBEROS_INCLUDE"
  fi

  AC_MSG_CHECKING([for certificate file])
  if test -f $openssl_certificate
  then
   AC_DEFINE_UNQUOTED(CERTF, "$openssl_certificate")
   AC_MSG_RESULT([$openssl_certificate found])
  else
   AC_MSG_RESULT([$openssl_certificate not found !!!])
  fi

  AC_MSG_CHECKING([for private key file])
  if test -f $openssl_privatekey
  then
   AC_DEFINE_UNQUOTED(KEYF, "$openssl_privatekey")
   AC_MSG_RESULT([$openssl_privatekey found])
  else
   AC_MSG_RESULT([$openssl_privatekey not found !!!])
  fi

  AC_DEFINE(WITH_SSL)
  ADD_SSL="pop_ssl.o"
else
  AC_MSG_RESULT(no)
fi
AC_SUBST(ADD_SSL)
```

Teraz można być pewnym, że wartości parametrów przekazanych do pliku konfiguracyjnego są prawidłowe i Teapop po skompilowaniu programem *make* będzie działał prawidłowo.

4.3.2 Zmiany w pliku Makefile.in

Chociaż zmiany w tym pliku polegają dosłownie na dopisaniu kilku znaków, to jednak bez tego nie byłaby możliwa kompilacja Teapop-a z SSL. Zmiany w Makefile.in ograniczają się do umieszczenia `@ADD_SSL@` w zmiennej `OBJS`, która określa, jakie pliki należy skompilować.

Linijka, w której została wprowadzona zmiana została wyróżniona wykrzyknikiem:

```
OBJS    = @ADD_FLOCK@ @ADD_LOCKF@ @ADD_MD5@ pop_auth.o pop_cmd_capa.o   \
          pop_cmd_dele.o pop_cmd_last.o pop_cmd_list.o pop_cmd_noop.o   \
          pop_cmd_retr.o pop_cmd_rset.o pop_cmd_stat.o pop_cmd_top.o    \
          pop_cmd_uidl.o pop_dele.o pop_dnld.o pop_file.o pop_hello.o   \
!         @ADD_JAVA@ @ADD_LDAP@ @ADD_SSL@ pop_lock.o pop_maildir.o      \
          pop_mbox.o @ADD_MYSQL@ pop_parse.o pop_passwd.o @ADD_PGSQL@   \
          pop_popsmtp.o pop_priv.o @ADD_DRAC@ @ADD_WHOSON@              \
          @ADD_POPAUTH_FILE@ pop_profil.o pop_signal.o pop_stat.o       \
          pop_strings.o pop_socket.o teapop.o version.o
```

Dodatek A

Uzgadnianie i kończenie połączenia TCP

Do ustanawiania i kończenia połączenia TCP jest wykorzystywany specjalny mechanizm zwany znacznikami (ang. *flags*). Znaczniki są dołączane do nagłówka TCP i każdy znacznik pełni inną funkcję w procesie połączenia TCP. Znaczenie poszczególnych znaczników zostało przedstawione w tabeli A.

Znacznik	**Opis**
SYN	Numery synchronizacji sekwencji. Używane w celu ustanowienia połączenia.
FIN	Nadawca zakończył połączenie. Używane w celu kończenia połączenia.
RST	Zerowanie połączenia.
PSH	Wysyłanie danych.
ACK	Potwierdzenie.
URG	Pilny.

Tabela A.1: Znaczniki wykorzystywane w połączeniach TCP

A.1 Uzgadnianie połączenia TCP

Nagłówek pakietu TCP zawiera różne znaczniki, które są konieczne do ustanowienia połączenia. Serwery wymieniają pakiety zawierające znaczniki, aby określić, kto otwiera połączenia, a kto je zamyka. Serwery muszą potwierdzić każdy znacznik, dlatego każdy z nich rozpoznaje, kiedy proces połączenia jest zakończony. Gdyby potwierdzenia nie były wymagane, przejęcie wciąż aktywnego połączenia byłoby znacznie prostsze dla hakerów, nawet wtedy, gdy oryginalny serwer uznałby takie połączenie za zakończone.

Do ustanowienia połączenia TCP są potrzebne trzy kroki. Poniższy schemat prezentuje w jaki sposób jest ustanawiane połączenie pomiędzy klientem a serwerem.

1. **Klient:** W nagłówku pakietu przesyłany jest znacznik SYN razem z numerem portu, na którym ma się odbyć komunikacja (jak na przykład port 110 serwera POP3). Przesyłany jest numer początkowy sekwencji klienta (ang. Initial Sequence Number ISN).
2. **Serwer:** Odpowiada własnym znacznikiem SYN i numerem ISN na port TCP klienta. Odpowiedź zawiera również ustawiony znacznik ACK będący potwierdzeniem na znacznik SYN klienta.
3. **Klient:** Odpowiada znacznikiem ACK czym potwierdza, że otrzymał znacznik SYN od serwera.

Po tej wymianie uzgadnianie dobiegło końca i połączenie TCP jest ustanowione. Teraz protokoły warstwy aplikacji, takie POP3, SMTP, HTTP mogą korzystać z tego połączenia TCP. Ten trzy stopniowy proces uzgadniania połączenia został zaprezentowany na rysunku A.1.

Rysunek A.1: Uzgadnianie połączenia TCP

A.2 Kończenie połączenia TCP

Gdy wymiana danych pomiędzy serwerem, a klientem dobiegła do końca, połączenie TCP należy zamknąć. Są cztery kroki, w których połączenie jest zamykane. Połączenie TCP może zostać zakończone przez oba hosty, w omówionym przypadku to serwer zakończy komunikację:

1. **Serwer:** Przesyła flagę FIN do klienta. Jest ona zwykle wysyłana jako rezultat zamknięcia aplikacji po stronie serwera.
2. **Klient:** Potwierdza, że połączenie będzie zakończone wysyłając znacznik ACK.
3. **Klient:** Przesyła flagę FIN do serwera.
4. **Serwer:** Odpowiada flagą ACK w celu potwierdzenia, że połączenie TCP jest zakończone.

Proces zamykania połączenia TCP jest pokazany na rysunku A.2.

Rysunek A.2: Kończenie połączenia TCP

Dodatek B

Funkcje i typy danych wykorzystane z biblioteki OpenSSL

B.1 Opis wykorzystanych funkcji z biblioteki OpenSSL

B.1.1 SSL_CTX_new

Składnia:

```
#include <openssl/ssl.h>

SSL_CTX *SSL_CTX_new(SSL_METHOD *method);
```

Opis:

Tworzy nowy obiekt `SSL_CTX` , w ramach, którego zostanie ustanowione połączenia TLS/SSL.

Zwracane wartości:

`NULL`
: Próba utworzenia nowego obiektu nie powiodła się. Należy sprawdzić stos błędów, aby dowiedzieć się, co było tego przyczyną.

Wskaźnik do obiektu `SSL_CTX`
: Zwracana wartość wskazuje na komórkę pamięci, w której ulokowany jest obiekt `SSL_CTX`.

B.1.2 SSL_CTX_free

Składnia: `#include <openssl/ssl.h>`

```
void SSL_CTX_free(SSL_CTX *ctx);
```

Opis:

`SSL_CTX_free()` usuwa object `SSL_CTX` na który wskazuje zmienna ctx. Funkcja wywołuje także procedury zwalniające pamięć, zajętą przez obiekty wykorzystane do utworzenia struktury `SSL_CTX`, a więc: certyfikaty i klucze, listę szyfrów, pamięć podręczną sesji, listę certyfikatów CA klienta.

Zwracane wartości:

Funkcja nie zwraca żadnych użytecznych wartości.

B.1.3 SSL_new

Składnia:

```
#include <openssl/ssl.h>

SSL *SSL_new(SSL_CTX *ctx);
```

Opis:

Tworzy nową strukturę `SSL` która jest potrzebna, aby przechowywać dane połączenia TLS/SSL. Nowa struktura dziedziczy ustawienia obiektu `SSL_CTX` na który wskazuje zmienna `ctx`, między innymi: metodę połączenia (SSLv2/SSLv3/TLSv1), opcje, weryfikację ustawień.

Zwracane wartości:

`NULL`
: Próba utworzenia nowej struktury `SSL` nie powiodła się. Należy sprawdzić stos błędów, aby znaleźć przyczynę.

Wskaźnik do struktury SSL
: Zwrócona wartość wskazuje na zaalokowaną strukturę `SSL`.

B.1.4 SSL_free

Składnia:

```
#include <openssl/ssl.h>

void SSL_free(SSL *ssl);
```

Opis:

Zwalnia pamięć użytą przez strukturę `SSL`, na którą wskazuje zmienna `ssl`.

Zwracane wartości:
Nie dostarcza diagnostycznej informacji.

B.1.5 SSL_load_error_strings

Składnia:

```
#include <openssl/ssl.h>

void SSL_load_error_strings(void);
```

Opis:
Rejestruje kody błędów dla wszystkich funkcji z bibliotek crypto oraz ssl. Funkcja ta powinna zostać wywołana przed generowaniem komunikatów o błędach.

Zwracane wartości:
Nie zwraca żadnej wartości.

B.1.6 ERR_peek_last_error

Składnia:

```
#include <openssl/err.h>

unsigned long ERR_peek_last_error(void);
```

Opis:
Zwraca ostatni kod błędu ze stosu błędów nie modyfikując go.

Zwracane wartości:
Ostatni kod błędu lub, 0 gdy nie było błędów na stosie.

B.1.7 ERR_error_string_n

Składnia:

```
#include <openssl/err.h>

char *ERR_error_string_n(unsigned long e, char *buf, size_t len);
```

Opis:
Wypisuje do bufora `buf`, komunikat o błędzie. Maksymalna długość komunikatu (włączając znak końca ciągu) nie może przekraczać `len` znaków, jeśli zajdzie taka konieczność informacja o błędzie zostanie obcięta

do.

Dla funkcji `ERR_error_string_n()`, `buf` nie może mieć wartości `NULL`.

Format komunikatu o błędzie jest następujący:

```
error:[kod błędu]:[nazwa biblioteki]:[nazwa funkcji]:[przyczyna]
```

`kod błędu` jest to 8-mio cyfrowa liczba szesnastkowa, `nazwa biblioteki`, `nazwa funkcji` i `przyczyna` są to napisy w kodzie ASCII.

Zwracane wartości:
Ponieważ przekazany parametr `buf` nie może mieć wartości NULL, więc funkcja zawsze zwraca `buf`.

B.1.8 SSLeay_add_ssl_algorithms

Składnia:

```
#include <openssl/ssl.h>

int SSLeay_add_ssl_algorithms(void);
```

Opis:
Zainicjowanie biblioteki SSL przez rejestrację algorytmów kryptograficznych. `SSL_library_init()` i `OpenSSL_add_ssl_algorithms()` są to synonimy dla `SSLeay_add_ssl_algorithms()`.

Zwracane wartości:
`SSLeay_add_ssl_algorithms()` zawsze zwraca 1.

B.1.9 SSLv23_client_method

Składnia:

```
#include <openssl/ssl.h>

SSL_METHOD *SSLv23_server_method(void)
```

Opis:
Komunikacja będzie odbywać się za pomocą wszystkich trzech protokołów. Funkcja zapewnia, że obsłużona zostanie komunikacja zarówno za pomocą SSLv2, SSLv3, jak i TLSv1.

Zwracane wartości:
Zwraca obiekt typu `SSL_METHOD`.

B.1.10 SSL_CTX_use_certificate_file

Składnia:

```
#include <openssl/ssl.h>

int SSL_use_certificate_file(SSL *ssl, const char *file, int type);
```

Opis:
Wczytuje certyfikat przechowywany w pliku do obiektu ctx. Przekazywany do funkcji parametr `type` powinien przyjmować jedną z następujących wartości: `SSL_FILETYPE_PEM`, `SSL\_FILETYPE\_ASN1`. Wartość parametru `type` określa format certyfikatu.

Zwracane wartości:
Gdy operacja się powiedzie funkcja zwraca 1, w przeciwnym razie należy sprawdzić stos błędów w celu znalezienia przyczyny.

B.1.11 SSL_CTX_use_PrivateKey_file

Składnia:

```
#include <openssl/ssl.h>

int SSL_CTX_use_PrivateKey_file(SSL_CTX *ctx, const char *file, int type);
```

Opis:
Dodaje pierwszy znaleziony klucz w pliku `file` do obiektu `ctx`. Format certyfikatu przekazany do funkcji przez parametr `type`, musi być jednym ze znanych typów: `SSL_FILETYPE_PEM` lub `SSL_FILETYPE_ASN1`.

Zwracane wartości:
Gdy operacja się powiedzie funkcja zwraca 1, w przeciwnym razie należy sprawdzić stos błędów w celu znalezienia przyczyny.

B.1.12 SSL_CTX_check_private_key

Składnia:

```
#include <openssl/ssl.h>

int SSL_check_private_key(SSL *ssl);
```

Opis:
Sprawdza spójność klucza prywatnego z odpowiednim świadectwem wczytanym do obiektu `ctx`.

Zwracane wartości:

Gdy operacja się powiedzie funkcja zwraca 1, w przeciwnym razie należy sprawdzić stos błędów w celu znalezienia przyczyny.

B.1.13 SSL_set_fd

Składnia:

```
#include <openssl/ssl.h>

int SSL_set_fd(SSL *ssl, int fd);
```

Opis:

Łączy object SSL wskazywany przez zmienną ssl z deskryptorem pliku `fd`.

Zwracane wartości:

0
: Operacja się nie powiodła się. Należy sprawdzić stos błędów żeby dowiedzieć się, dlaczego.

1
: Operacja zakończyła się sukcesem.

B.1.14 SSL_accept

Składnia:

```
#include <openssl/ssl.h>

int SSL_accept(SSL *ssl);
```

Opis:

Po wywołaniu funkcji `SSL_accept`, serwer czeka na połączenie TLS/SSL od klienta, aby zainicjować uzgadnianie połączenia TLS/SSL. Kanał komunikacyjny musi już być ustawiony i dowiązany do struktury `ssl`.

Zwracane wartości:

1
: Uzgadnianie połączenia TLS/SSL zostało zakończone pomyślnie.

0
: Uzgadnianie połączenia TLS/SSL nie powiodło się i połączenie TLS/SSL zostało poprawnie zamknięte.

<0
: Uzgadnianie połączenia TLS/SSL nie powiodło się, ponieważ wystąpił błąd krytyczny na poziomie protokołu lub połączenie zostało przerwane. Połączenie TLS/SSL nie zostało zamknięte poprawnie.

B.1.15 SSL_read

Składnia:

```
#include <openssl/ssl.h>

int SSL_read(SSL *ssl, void *buf, int num);
```

Opis:

Pobiera od obiektu `ssl` reprezentującego połączenie, dane otrzymane od drugiego komputera uczestniczącego w połączeniu i zapisuje je do bufora `buf`

Zwracane wartości:

>0
: Operacja przebiegła pomyślnie i zwracana jest liczba odczytanych przez funkcję bajtów z połączenia TLS/SSL.

0
: Operacja czytania nie przebiegła pomyślnie. Aby znaleźć przyczynę wystąpienia błędu należy wywołać funkcję `SSL_get_error()` i przeanalizować zwróconą wartość.

<0
: Operacja czytania nie powiodła się. Żeby poznać przyczynę należy wywołać funkcję `SSL\_get\_error()`, która zwraca kod błędu dla operacji wejścia/wyjścia połączenia TLS/SSL.

B.1.16 SSL_write

Składnia:

```
#include <openssl/ssl.h>

int SSL_write(SSL *ssl, const void *buf, int num);
```

Opis:

Przekazuje do obiektu `ssl`, a tym samym wysyła do drugiej strony połączenia liczbę `num` bajtów z bufora `buf`.

Zwracane wartości:

>0
: Operacja przebiegła pomyślnie i zwracana jest liczba wysłanych przez funkcję bajtów do połączenia TLS/SSL.

0
: Operacja zapisu nie powiodła. Prawdopodobnie przyczyną błędu było zamknięte połączenie. Szczegółowych informacji dostarczy wywołanie `SSL_get_error()`. Dowiemy się czy wystąpił błąd i czy połączenie zostało zamknięte poprawnie.
W protokole SSLv2 możliwe jest jedynie wykrycie, czy wystąpienie błędu ma związek z zamknięciem połączenia. Nie można natomiast sprawdzić, dlaczego doszło do zamknięcia połączenia.

<0
: Operacja zapisu nie powiodła się. Wywołanie funkcji `SSL_get_error()`, oraz interpretacja zwróconej wartości pomoże w znalezieniu przyczyny.

B.2 Typy danych

B.2.1 Definicja struktury `SSL`

```
struct ssl_st
        {
        /* protocol version
         * (one of SSL2_VERSION, SSL3_VERSION, TLS1_VERSION)
         */
        int version;
        int type; /* SSL_ST_CONNECT or SSL_ST_ACCEPT */

        SSL_METHOD *method; /* SSLv3 */

        /* There are 2 BIO's even though they are normally both the
         * same.  This is so data can be read and written to different
         * handlers */

#ifndef OPENSSL_NO_BIO
        BIO *rbio; /* used by SSL_read */
        BIO *wbio; /* used by SSL_write */
        BIO *bbio; /* used during session-id reuse to concatenate
                    * messages */
#else
        char *rbio; /* used by SSL_read */
        char *wbio; /* used by SSL_write */
        char *bbio;
#endif
        /* This holds a variable that indicates what we were doing
         * when a 0 or -1 is returned.  This is needed for
         * non-blocking IO so we know what request needs re-doing when
         * in SSL_accept or SSL_connect */
        int rwstate;

        /* true when we are actually in SSL_accept() or SSL_connect() */
        int in_handshake;
```

```
        int (*handshake_func)();

        /* Imagine that here's a boolean member "init" that is
         * switched as soon as SSL_set_{accept/connect}_state
         * is called for the first time, so that "state" and
         * "handshake_func" are properly initialized.  But as
         * handshake_func is == 0 until then, we use this
         * test instead of an "init" member.
         */

        int server;     /* are we the server side? - mostly used by SSL_clear*/

        int new_session;/* 1 if we are to use a new session.
                         * 2 if we are a server and are inside a handshake
                         *   (i.e. not just sending a HelloRequest)
                         * NB: For servers, the 'new' session may actually be a previously
                         * cached session or even the previous session unless
                         * SSL_OP_NO_SESSION_RESUMPTION_ON_RENEGOTIATION is set */
        int quiet_shutdown;/* don't send shutdown packets */
        int shutdown;   /* we have shut things down, 0x01 sent, 0x02
                         * for received */
        int state;      /* where we are */
        int rstate;     /* where we are when reading */

        BUF_MEM *init_buf;      /* buffer used during init */
        void *init_msg;         /* pointer to handshake message body, set by ssl3_get_message() */
        int init_num;           /* amount read/written */
        int init_off;           /* amount read/written */

        /* used internally to point at a raw packet */
        unsigned char *packet;
        unsigned int packet_length;

        struct ssl2_state_st *s2; /* SSLv2 variables */
        struct ssl3_state_st *s3; /* SSLv3 variables */

        int read_ahead;         /* Read as many input bytes as possible
                                 * (for non-blocking reads) */

        /* callback that allows applications to peek at protocol messages */
        void (*msg_callback)(int write_p, int version, int content_type, const void *buf, size_t len, SSL *ssl, void *ar
        void *msg_callback_arg;

        int hit;                /* reusing a previous session */

        int purpose;            /* Purpose setting */
        int trust;              /* Trust setting */

        /* crypto */
        STACK_OF(SSL_CIPHER) *cipher_list;
        STACK_OF(SSL_CIPHER) *cipher_list_by_id;

        /* These are the ones being used, the ones in SSL_SESSION are
         * the ones to be 'copied' into these ones */

        EVP_CIPHER_CTX *enc_read_ctx;           /* cryptographic state */
        const EVP_MD *read_hash;                /* used for mac generation */
#ifndef OPENSSL_NO_COMP
        COMP_CTX *expand;                       /* uncompress */
#else
        char *expand;
#endif

        EVP_CIPHER_CTX *enc_write_ctx;          /* cryptographic state */
        const EVP_MD *write_hash;               /* used for mac generation */
#ifndef OPENSSL_NO_COMP
```

```
        COMP_CTX *compress;                     /* compression */
#else
        char *compress;
#endif

        /* session info */

        /* client cert? */
        /* This is used to hold the server certificate used */
        struct cert_st /* CERT */ *cert;

        /* the session_id_context is used to ensure sessions are only reused
         * in the appropriate context */
        unsigned int sid_ctx_length;
        unsigned char sid_ctx[SSL_MAX_SID_CTX_LENGTH];

        /* This can also be in the session once a session is established */
        SSL_SESSION *session;

        /* Default generate session ID callback. */
        GEN_SESSION_CB generate_session_id;

        /* Used in SSL2 and SSL3 */
        int verify_mode;        /* 0 don't care about verify failure.
                                 * 1 fail if verify fails */
        int verify_depth;
        int (*verify_callback)(int ok,X509_STORE_CTX *ctx); /* fail if callback returns 0 */

        void (*info_callback)(const SSL *ssl,int type,int val); /* optional informational callback */

        int error;              /* error bytes to be written */
        int error_code;         /* actual code */

#ifndef OPENSSL_NO_KRB5
        KSSL_CTX *kssl_ctx;     /* Kerberos 5 context */
#endif  /* OPENSSL_NO_KRB5 */

        SSL_CTX *ctx;
        /* set this flag to 1 and a sleep(1) is put into all SSL_read()
         * and SSL_write() calls, good for nbio debuging :-) */
        int debug;

        /* extra application data */
        long verify_result;
        CRYPTO_EX_DATA ex_data;

        /* for server side, keep the list of CA_dn we can use */
        STACK_OF(X509_NAME) *client_CA;

        int references;
        unsigned long options; /* protocol behaviour */
        unsigned long mode; /* API behaviour */
        long max_cert_list;
        int first_packet;
        int client_version;     /* what was passed, used for
                                 * SSLv3/TLS rollback check */
        };
```

B.2.2 Definicja struktury SSL_CTX

```
struct ssl_ctx_st
        {
        SSL_METHOD *method;

        STACK_OF(SSL_CIPHER) *cipher_list;
```

```
/* same as above but sorted for lookup */
STACK_OF(SSL_CIPHER) *cipher_list_by_id;

struct x509_store_st /* X509_STORE */ *cert_store;
struct lhash_st /* LHASH */ *sessions;  /* a set of SSL_SESSIONs */
/* Most session-ids that will be cached, default is
 * SSL_SESSION_CACHE_MAX_SIZE_DEFAULT. 0 is unlimited. */
unsigned long session_cache_size;
struct ssl_session_st *session_cache_head;
struct ssl_session_st *session_cache_tail;

/* This can have one of 2 values, ored together,
 * SSL_SESS_CACHE_CLIENT,
 * SSL_SESS_CACHE_SERVER,
 * Default is SSL_SESSION_CACHE_SERVER, which means only
 * SSL_accept which cache SSL_SESSIONS. */
int session_cache_mode;

/* If timeout is not 0, it is the default timeout value set
 * when SSL_new() is called.  This has been put in to make
 * life easier to set things up */
long session_timeout;

/* If this callback is not null, it will be called each
 * time a session id is added to the cache.  If this function
 * returns 1, it means that the callback will do a
 * SSL_SESSION_free() when it has finished using it.  Otherwise,
 * on 0, it means the callback has finished with it.
 * If remove_session_cb is not null, it will be called when
 * a session-id is removed from the cache.  After the call,
 * OpenSSL will SSL_SESSION_free() it. */
int (*new_session_cb)(struct ssl_st *ssl,SSL_SESSION *sess);
void (*remove_session_cb)(struct ssl_ctx_st *ctx,SSL_SESSION *sess);
SSL_SESSION *(*get_session_cb)(struct ssl_st *ssl,
        unsigned char *data,int len,int *copy);

struct
        {
        int sess_connect;       /* SSL new conn - started */
        int sess_connect_renegotiate;/* SSL reneg - requested */
        int sess_connect_good;  /* SSL new conne/reneg - finished */
        int sess_accept;        /* SSL new accept - started */
        int sess_accept_renegotiate;/* SSL reneg - requested */
        int sess_accept_good;   /* SSL accept/reneg - finished */
        int sess_miss;          /* session lookup misses  */
        int sess_timeout;       /* reuse attempt on timeouted session */
        int sess_cache_full;    /* session removed due to full cache */
        int sess_hit;           /* session reuse actually done */
        int sess_cb_hit;        /* session-id that was not
                                 * in the cache was
                                 * passed back via the callback.  This
                                 * indicates that the application is
                                 * supplying session-id's from other
                                 * processes - spooky :-) */
        } stats;
```

Dodatek C

Definicja funkcji FIND_OPENSSL

```
AC_DEFUN(FIND_OPENSSL, [
  incs="$1"
  libs="$2"
  case "$incs---$libs" in
    ---)
      for d in /usr/ssl/include /usr/local/ssl/include /usr/include \
/usr/include/ssl /opt/ssl/include /opt/openssl/include \
/usr/local/ssl/include /usr/local/include ; do
       if test -f $d/openssl/ssl.h  ; then
         OPENSSL_INCLUDE=-I$d
       fi
      done

      for d in /usr/ssl/lib /usr/local/ssl/lib /usr/lib/openssl \
/usr/lib /usr/lib64 /opt/ssl/lib /opt/openssl/lib /usr/local/lib/ ; do
      if test -f $d/libssl.a || test -f $d/libssl.so || test -f $d/libssl.dylib ; then
        OPENSSL_LIB=$d
      fi
      done
      ;;
    ---* | *---)
      AC_MSG_ERROR([if either 'includes' or 'libs' is specified, both must be specified])
      ;;
    * )
      if test -f $incs/openssl/ssl.h  ; then
        OPENSSL_INCLUDE=-I$incs
      fi
      if test -f $libs/libssl.a || test -f $libs/libssl.so || test -f $libs/libssl.dylib ; then
        OPENSSL_LIB=$libs
      fi
      ;;
  esac

  # On RedHat 9 we need kerberos to compile openssl
  for d in /usr/kerberos/include
  do
   if test -f $d/krb5.h  ; then
     OPENSSL_KERBEROS_INCLUDE="$d"
   fi
  done


 if test -z "$OPENSSL_LIB" -o -z "$OPENSSL_INCLUDE" ; then
   echo "Could not find an installation of OpenSSL"
```

```
   if test -n "$OPENSSL_LIB" ; then
    if test "$IS_LINUX" = "true"; then
      echo "Looks like you've forgotten to install OpenSSL development RPM"
    fi
   fi
  exit 1
 fi
])
```

Spis tabel

Spis rysunków

Spis literatury

[1] KERNIGHAN B.W., RITCHIE D.M., *Język C*, Wydawnictwa Naukowo-Techniczne,

[2] Strona domowa serwera Teapop,
HTTP://WWW.TOONTOWN.ORG/TEAPOP/,

[3] Strona projektu OpenSSL,
HTTP://WWW.OPENSSL.ORG/,

[4] Myers, J. and M. Rose, *Post Office Protocol – Version 3*, RFC 1939, May 1996,

[5] Rivest, R., *The MD5 Message-Digest Algorithm*, RFC 1321, MIT Laboratory for Computer Science, April 1992,

[6] Myers, J., *POP3 AUTHentication command*, RFC 1734, Carnegie Mellon, December 1994,

[7] Housley, R., *Triple-DES and RC2 Key Wrapping*, RFC3217, RSA Laboratories December 2001,

[8] *TRANSMISSION CONTROL PROTOCOL* , RFC793, Information Sciences Institute University of Southern California, September 1981

[9] Matematyczne podstawy algorytmów kryptograficznych,
http://www.bezpieczenstwoit.pl/Artykuly/Kryptografia

[10] Narzędzia do MD5,
http://www.fourmilab.ch/md5

[11] Centrum certyfikacji dla ZUS-u,
http://www.cc.unet.pl/

[12] Firmowa strona autorów szyfru RSA,
http://www.rsasecurity.com/

[13] Strona domowa projektu PGP (Pretty Good Privicy),
http://www.pgp.com/,

[14] Numery portów IP, od 0 do 999,
http://www.networksorcery.com/enp/protocol/ip/ports00000.htm,

[15] Dokumentacja narzędzi: autoconf i automake,
http://www.gnu.org/manual/,