22.04.2022

Na dzisiejszych zajęciach zapoznamy się z podstawowymi pojęciami związanymi z szyfrowaniem transmisji przy pomocy TLS

- Przypomnij sobie materiał z wykładu o kryptpgrafii
- Jeżeli pracujesz na własnym systemie, zainstaluj narzędzie `openssl` (w SPK jest ono dostępne). Jest to bardzo użyteczne narzędzie tworzone w ramach projektu OpenSSL (razem z biblioteką libssl dla C) które pozwala m. in. na generowanie kryptograficznych kluczy i certyfikatów, szyfrowanie nimi plików itp.
- Pobierz paczkę `serwer.zip` dołączoną do dzisiejszych zajęć i zapoznaj się z prostym serwerem HTTP(S) `serwer_https.py`. Do jego uruchomienia w trybie HTTPS potrzebne będzie wygenerowanie certyfikatu (`server.crt`) oraz klucza prywatnego (`server.key`).
- Wygeneruj klucz oraz certyfikat według poniższego przepisu:

  Najpierw generujemy klucz prywatny oraz certyfikat "Autorytetu"/"Urzędu" CA, który podpisze [1] certyfikat naszego serwera:

  ```
  openssl genrsa -des3 -out my-root.key 2048
  openssl req -x509 -new -nodes -key my-root.key -sha256 -days 1825 -out my-root.pem
  ```

  Następnie generujemy klucz prywatny dla naszego serwera:

  ```
  openssl genrsa -out server.key 2048
  ```

  Używając wygenerowanego klucza oraz pliku konfiguracyjnego `my-server.ext` z paczki `serwer.zip`, generujemy *Certificate Signing Request*:

  ```
  openssl req -new -key server.key -out server.csr
  ```

  Powstały plik CSR normalnie wysłalibyśmy do urzędu CA. Na potrzeby przykładu podpisujemy sobie certyfikat sami:

  ```
  openssl x509 -req -in server.csr -CA my-root.pem -CAkey my-root.key \
    -CAcreateserial -out server.crt -days 365 -sha256 -extfile my-server.ext
  ```

  Mamy teraz plik certyfikatu `server.crt`. Zakładając, że ufamy "urzędowi" reprezentowanemu przez nasz certyfikat `my-root.pem`, możemy przy pomocy OpenSSL sprawdzić ważność certyfikatu serwera:

  ```
  openssl verify -verbose -CAfile my-root.pem server.crt
  ```

- Sprawdź, gdzie w Twoim systemie operacyjnym trzymane są certyfikaty zaufanych urzędów CA.
- (*nieobowiązkowe*) Uruchom serwer w domyślnym trybie HTTP (bez szyfrowania) i wyślij do niego zapytanie przy pomocy narzędzia `curl`. Jednocześnie przy pomocy Wireshark-a "podejrzyj" transmisję i sprawdź czy da się odczytać zawartość zapytania i odpowiedzi HTTP.
- Pakiet `ssl` z Pythona użyty w programie `serwer_https.py` spodziewa się pojedynczego pliku PEM zawierającego zarówno certyfikat oraz klucz prywatny serwera. Trzeba więc "skleić" obydwa pliki w jeden na przykład przy pomocy `cat`:

  ```
  cat server.crt server.key > server_key_and_cert.pem
  ```

Uruchom serwer w trybie HTTPS:

```
python3 serwer_https.py -s -p <numer_portu>
```

Wykonaj przy pomocy `curl` zapytanie pod adres HTTPS (`https://localhost:<nunmer_portu>`). Co się dzieje? Jak skłonić `curl`-a, żeby połączył się z takim serwerem?

- (*nieobowiązkowe*) Dla ćwiczenia jak powyżej podejrzyj komunikację pomiędzy `curl`-em a serwerem przy pomocy Wiresharka. Czy da się przeczytać zawartość zapytania i odpowiedzi?
- Jeżeli pracujesz na własnym systemie, spróbuj otworzyć adres `https://localhost:<nunmer_portu>` w przeglądarce internetowej. Co się wtedy dzieje?
- Znajdź sposób na ściągnięcie i zapisanie do pliku certyfikatu SSL naszego serwera. Można to zrobić na przykład przy pomocy pakietu `ssl` w Pythonie albo używając wspomnianego wcześniej narzędzia `openssl`. Porównaj ściągnięty certyfikat z wygenerowanym wcześniej plikiem `server.crt`.

  Ponownie zweryfikuj certyfikat (tym razem ten otrzymany od serwera) przy pomocy `openssl verify`. Taka weryfikacja odbywa się przy pomocy systemowej (albo dołączonej do przeglądarki) bazy zaufanych urzędów CA za każdym razem kiedy nawiązujemy połączenie HTTPS.

  Wykonaj zapytanie do testowego serwera przy pomocy `curl`-a podając mu ścieżkę do certyfikatu `my-root.pem` w celu umożliwienia weryfikacji.
- Wyodrębnij klucz publiczny serwera ze ściągniętego certyfikatu:

  ```
  openssl x509 -pubkey -noout -in plik_certyfikatu.crt > klucz_publiczny_serwera.pem
  ```
- Utwórz plik tekstowy z dowolną wiadomością do zakodowania. Zakoduj ją przy pomocy klucza publicznego serwera używając narzędzia openssl:

  ```
  openssl rsautl -encrypt -inkey klucz_pub_serwera.pem -pubin -in wiadomosc.txt \
  -out zakodowana.bin
  ```

  Spróbuj odpowiednio zdekodować stworzoną wiadomość przy pomocy klucza prywatnego serwera (server.key). Spróbuj wykonać to samo ćwiczenie w drugą stronę - kodując przy pomocy klucza prywatnego serwera i dekodując przy pomocy klucza publicznego.
- Używając wybranego języka i biblioteki HTTP, stwórz klienta łączącego się z testowym serwerem. Będzie to wymagało dodania certyfikatu serwera do bazy zaufanych certyfikatów (większość bibliotek pozwala na wskazanie bazy certyfikatów, które mają być traktowane jako zaufane).

---

[1]: W rzeczywistości podpisanie certyfikatu byłoby płatną usługą ze strony jednego z zaufanych urzędów. Ostatnio można również skorzystać z Let's Encrypt