Bezpieczeństwo Sieci Komputerowych

Laboratorium 3: Bezpieczne usługi sieciowe, wirtualne sieci prywatne
Instrukcja do instrukcji

Kilka informacji wstępnych:
W instrukcji do laboratorium niektóre zagadnienia zostały trochę słabo opisane, więc spróbowałem zebrałem tutaj różne informacje, które mogą pomóc w wykonaniu wszystkich ćwiczeń i napisaniu sprawozdania. Do kilku zagadnień dopisałem przy okazji trochę informacji, które nie są związane bezpośrednio z programem laboratorium, ale mogą przydać się Wam w praktyce – są one oznaczone podkreślonym *w praktyce* na początku tekstu.

W opisie zastosowałem typową konwencję oznaczania komend wykonywanych w linuksowym/uniksowym terminalu:
$ – tak oznaczoną komendę można wykonać z uprawnieniami zwykłego użytkownika
# – tak oznaczoną komendę trzeba wykonać z uprawnieniami roota (polecenie *sudo*)
Ponieważ do większości komend wykonywanych na tych zajęciach potrzebne są uprawnienia roota, a w oryginalnej instrukcji nie jest to dobrze zaznaczone, na początku polecam użyć komendy:

```
$ sudo su
```

I wszystkie kolejne wykonywać już od razu z uprawnieniami administracyjnymi – zaoszczędzi to Wam trochę pisania. Do edycji plików z poziomu terminala (w kilku miejscach jest taka potrzeba) polecam program *nano*. Edytując pliki konfiguracyjne w katalogu */etc* trzeba oczywiście mieć uprawnienia roota.
W oryginalnej instrukcji stosowane są 3 różne sposoby restartowania usług (demonów), z czego jedna jest w tym momencie w Ubuntu przestarzała i zachowana tylko dla kompatybilności, druga nie jest natywną (z tego co słyszałem wywodzi się z Red Hata). Tutaj dla porządku będę trzymaj się jednej, najnowszej (systemd).

Punkt 1.
W pierwszym punkcie mamy zainstalować i uruchomić w podstawowej konfiguracji serwer WWW oraz zmienić stronę na własną - tu specjalnych trudności nie ma, standardowo:

```
# apt install apache2
```

Strona serwera Apache domyślnie na Ubuntu/Debianie znajduje się w /*var/www/html*. Jak już serwer będzie działał, trzeba sprawdzić poufność transmisji – czyli uruchamiamy Wiresharka, łączymy się z drugiego komputera z naszym serwerem i oglądamy przechwycone pakiety. Sam kod HTML może nie być dobrze widoczny, jeżeli na serwerze działa kompresja, ale wszystkie żądania HTTP będą wysyłane jawnym tekstem.

Tutaj warto jeszcze wspomnieć polecenie, którym na współczesnych wersjach Ubuntu łatwo można sprawdzić adres IP:

```
$ ip addr show
```

Punkt 2.
Konfiguracja HTTPS – instalujemy OpenSSL i uruchamiamy odpowiedni moduł Apache:

```
# apt install openssl
# a2enmod ssl
# systemctl restart apache2
```

Standardowo Apache będzie nasłuchiwał na porcie 443, ale dla pewności można jeszcze zajrzeć rzeczywiście do /e*tc/apache2/ports.conf*. Generujemy certyfikat tak jak w instrukcji i konfigurujemy stronę HTTPS – tu nie trzeba tworzyć nowego pliku z konfiguracją, najprościej będzie w domyślnym */etc/apache2/sites-available/default-ssl.conf* ustawić nasze ścieżki do certyfikatu i klucza w *SSLCertificateFile* i *SSLCertificateKeyFile*. Jak to zrobimy, musimy uruchomić stronę i przeładować serwer:

```
# a2ensite default-ssl
# systemctl reload apache2
```

Jak wszystko pójdzie dobrze, powinno dać się wejść na stronę po HTTPS z drugiego komputera.

Punkt 3.
Tu należy zbadać parę szczegółów połączenia. Z algorytmami i szczegółami certyfikatu sprawa jest raczej oczywista, nie ma co wyjaśniać – natomiast przydałoby się parę słów dlaczego Firefox wyświetla komunikat, że połączenie nie jest bezpieczne.
Chodzi o to, że wykorzystaliśmy certyfikat samodzielnie podpisany (self-signed) – w związku z tym przeglądarka nie jest pewna, kto ten nasz certyfikat wystawił i nie wie, czy można ufać co do jego autentyczności. Inaczej mówiąc, taki certyfikat zapewnia poufność połączenia, ale nie zapewnia uwierzytelnienia serwera (Firefox nie może potwierdzić, że osoba czy instytucja podana w certyfikacie rzeczywiście jest administratorem tego serwera).

Kiedy to będzie możliwe? Przeglądarka musi wiedzieć, że klucz prywatny użyty do podpisania certyfikatu należy do kogoś, komu można ufać i nie podpisze on certyfikatu dla naszej domeny tak po prostu, byle komu. Do ustalenia tego służy łańcuch zaufania (chain of trust) – potrzebny nam certyfikat jest podpisywany za pomocą jakiegoś klucza, dla którego też został wystawiony certyfikat i też został on podpisany za pomocą jakiegoś klucza. Takich poziomów w górę może być wiele, ale ważne jest, że w końcu na samej górze znajdzie się certyfikat jakiegoś urzędu certyfikacji (Certificate Authority) podpisany samodzielnie. Każda przeglądarka ma bazę takich certyfikatów najwyższego poziomu (tzw. root CA), których wystawcom ufa. Jeżeli używany przez nas certyfikat odwołuje się w końcu do jednego z nich, przeglądarka uzna połączenie za bezpieczne. W naszym przypadku łańcuch zaufania składa się oczywiście tylko z jednego poziomu (self-signed).

Co możemy zrobić, żeby certyfikat dla naszej strony był zaufany? Są dwie opcje:

- Możemy zaimportować do zaufanej bazy przeglądarki nasz certyfikat najwyższego poziomu (Firefox: Preferencje > Prywatność i bezpieczeństwo > Certyfikaty > Wyświetl certyfikaty > Organy certyfikacji > Importuj). W przypadku z laborek byłby to ten sam certyfikat, co używany przez serwer, niekoniecznie ma to sens. Ale może to być przydatne w innej sytuacji, np. w firmie można zaimportować certyfikat własnego urzędu certyfikacji (root CA) używany do podpisywania innych certyfikatów stosowanych w sieci wewnętrznej.
- W praktyce – uzyskać certyfikat powszechnie uznawany za zaufany (i to jest jedyna możliwość, jak stawiamy stronę „dla świata"). Bezpłatnie można taki dostać od Let's Encrypt (https://letsencrypt.org) – wszystko odbywa się zdalnie, weryfikacja polega na udowodnieniu przez odpowiednie skonfigurowanie serwera, że to my jesteśmy jego

administratorami. Certyfikat jest ważny krótko (180 dni), ale bardzo łatwo można zautomatyzować jego odświeżanie – służy do tego pakiet *certbot*, na Debianie/Ubuntu dostępny w repozytoriach systemowych.

Jak teraz przyjrzymy się pakietom w Wiresharku, zbyt wiele już nie zobaczymy. Jawnie powinien zostać przesłany sam certyfikat (wtedy jeszcze nie ma możliwości szyfrowania), ale to nie problem – ponieważ zawiera oczywiście tylko klucz publiczny. Później przeglądarka może już zacząć szyfrowaną łączność z serwerem, wykorzystując jego klucz publiczny.

Punkt 4.
Tutaj będziemy musieli zrobić trochę rozeznania jakie zestawy szyfrów są obsługiwane przez serwer i przez przeglądarkę, a później wymusić użycie jakiegoś innego niż domyślny. Niezawodne zestawy szyfrów akceptowane przez Firefoxa są do znalezienia na tej stronie (sekcja *How can I create an SSL server which accepts strong encryption only?*):
https://httpd.apache.org/docs/trunk/ssl/ssl_howto.html
Ustawić trzeba SSLCipherSuite w pliku */etc/apache2/mods-available/ssl.conf*, można też dopisać *SSLHonorCipherOrder*. Jeżeli nawet nie każdy, to któryś z podanych tam zestawów na pewno zadziała.

Punkt 5.
Jeżeli nie pamiętacie składni polecenia *useradd*, na Ubuntu jest skrypt do prostego tworzenia kont użytkowników, który o wszystkie dane sam pyta, w tym o hasło (z tym, że to dotyczy samego Ubuntu, na czystym Debianie już go nie będzie):

```
# adduser nazwa_konta
```

Za pomocą *useradd* trzeba by najpierw utworzyć konto (parametr *-m* tworzy automatycznie katalog domowy), a później ustawić hasło:

```
# useradd -m nazwa_konta
# passwd nazwa_konta
```

Punkt 6.
Instalacja i uruchomienie serwera (demona) telnet:

```
# apt install telnetd
# systemctl restart openbsd-inetd
```

Standardowo telnet działa na porcie 23, po jego zmianie w */etc/services* należy zrestartować jeszcze raz demona podanym przed chwilą poleceniem. I oczywiście telnet nie zapewnia żadnego szyfrowania, czyli za pomocą *Follow TCP Stream* w Wiresharku spokojnie możemy podejrzeć całą wymianę tekstu z serwerem, z hasłami włącznie.

Punkt 7.
Instalacja i uruchomienie serwera SSH:

```
# apt install openssh-server
```

Plik konfiguracyjny serwera SSH to /etc/ssh/sshd_conf. Domyślny port to 22, ustawienie jest gdzieś na początku i jest zakomentowane, musimy odkomentować i zmienić na jakiś spoza zakresu 0-1023 (well-known ports). Oprócz tego mamy zmienić jakieś 2 inne ustawienia, możemy np. liczbę nieudanych prób logowania po których nastąpi rozłączenie (*MaxAuthTries*) lub wiadomość

tekstową wyświetlaną przed logowaniem (*Banner* – tam trzeba podać ścieżkę do pliku z wiadomością). Po ustawieniu wszystkiego trzeba zrobić restart demona SSH:

```
# systemctl restart sshd
```

W praktyce taka zmiana portu SSH jak w tym ćwiczeniu pozwala dość ładnie wyciąć próby logowania się na nasz serwer przez wiele botów próbujących łamać hasła metodą brutalną, ale nie zawsze możemy ją zastosować. Na Linuksie do zabezpieczenia serwera przed brute-force'owaniem haseł polecam jeszcze zainstalować i skonfigurować program fail2ban, który monitoruje logi demona SSH i po kilku nieudanych próbach logowania banuje na określony czas źródłowy adres IP.

W przypadku SSH potwierdzenie autentyczności serwera odbywa się przez sprawdzenie przy łączeniu, czy klucz hosta (generowany przy instalacji demona) zgadza się z używanym poprzednio. Ponieważ przy pierwszym połączeniu klient nie zna serwera, wyświetla po prostu odcisk klucza hosta i pyta, czy należy mu ufać – nie jest to w żaden sposób zautomatyzowane, jeżeli użytkownik potrzebuje uwierzytelnić serwer, to zadanie należy już do niego. W wielu przypadkach nie jest to w ogóle potrzebne (np. jak łączymy się w domu po SSH z właśnie uruchomionym Raspberry Pi), a jeżeli musimy jednak mieć pewność (chociażby jakieś połączenie z zewnątrz z firmowym serwerem, na którym będziemy pracować na ważnych danych), powinniśmy porównać wyświetlony u nas skrót klucza ze skrótem podanym przez administratora serwera w wystarczająco bezpieczny dla nas sposób (na stronie internetowej/w rozmowie telefonicznej/osobiście, itd.).

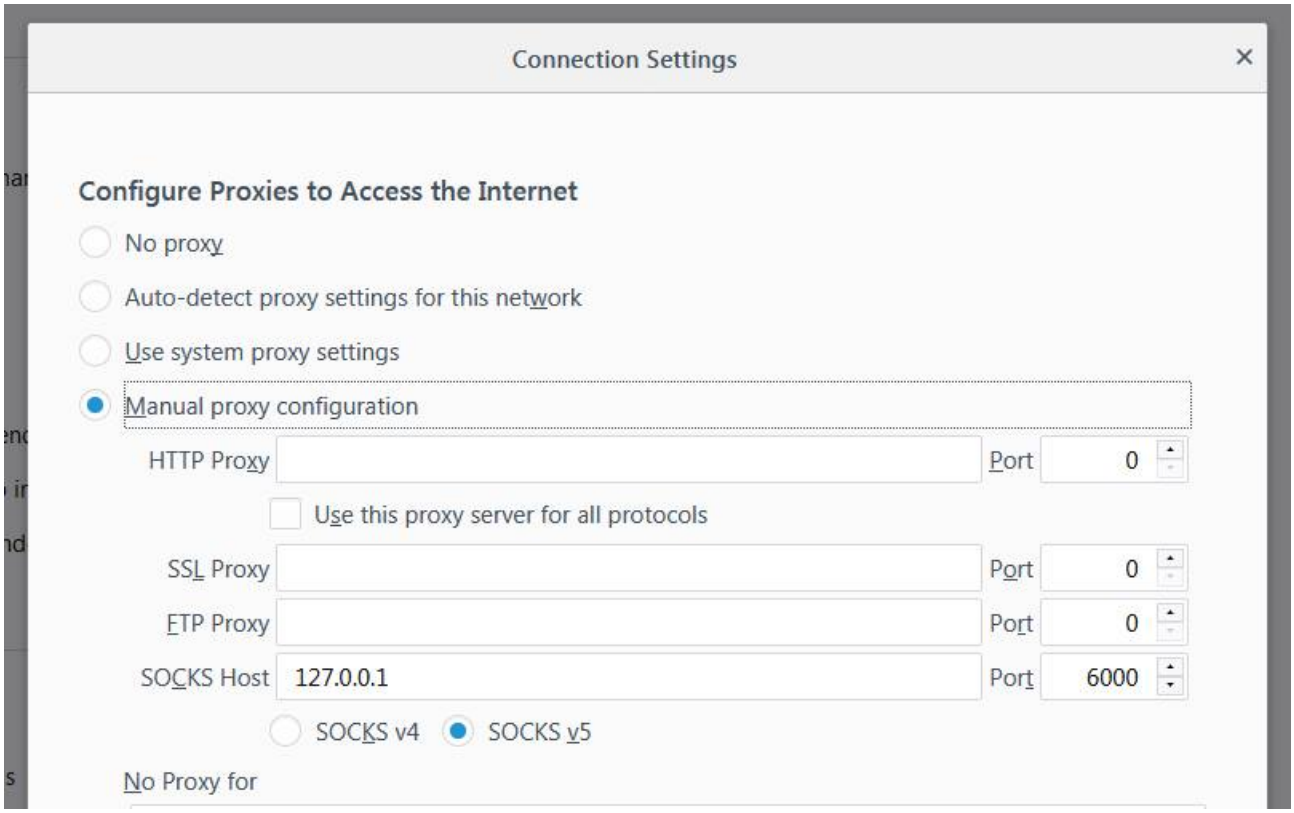
Na laborkach dr Markowski prosi o wyświetlenie odcisku klucza po stronie serwera. Istotne jest, żeby wygenerować go tak, jak po drugiej stronie wyświetla Putty i z odpowiedniego klucza. Służy do tego polecenie:

```
$ ssh-keygen -l -E md5 -f /etc/ssh/ssh_host_ed25519_key.pub
```

Standardowo w tej chwili wykorzystywany jest algorytm ed25519 i jego klucz, ale gdyby z jakiegoś powodu Putty powołał się na inny (np. rsa, chociaż nie powinien, jest słabszy) – należy wygenerować odcisk innego. Wtedy po prostu wylistujcie zawartość /etc/ssh (polecenie ls) i znajdźcie odpowiedni plik klucza.

W praktyce z kluczem hosta wiąże się jeszcze jedna rzecz. Ponieważ jest on generowany raz i później sprawdzany przy łączeniu, w przypadku nowej instalacji systemu operacyjnego na serwerze dobrym pomysłem jest skopiowanie starych kluczy z katalogu */etc/ssh* na poprzedniej instalacji i zastąpienie nimi kluczy hosta wygenerowanych przy nowej instalacji.

Punkt 8.
W tym punkcie mamy do zestawienia prosty tunel SSH, żeby ominąć blokadę portu 80 (HTTP) w sieci, do której mamy podłączyć jeden z komputerów. Po stronie naszego serwera nic nie musimy konfigurować, wszystko trzeba zrobić po stronie klienta. Konfiguracja proxy w Firefoksie – Preferencje > Ogólne > Sieć > Ustawienia – powinna wyglądać tak (na następnej stronie):



Ilustracja 1: Konfiguracja proxy w przeglądarce Firefox

Konfiguracja Putty jest opisana dobrze w instrukcji, ustawiamy tylko S*ource port* na 6000 i zaznaczamy opcję *Dynamic*, pole *Destination* ma być puste. Z tymi ustawieniami łączymy się z serwerem i logujemy – i tunel powinien działać.

I to chyba wszystko, powodzenia!

Jan Potocki
Wrocław, 15.11.2018