# 10. Bezpieczeństwo aplikacji i usług sieciowych

Bieżący moduł przedstawia wybrane zagadnienia bezpieczeństwa dotyczące aplikacji użytkowych i usług sieciowych. Najpierw omówiony zostanie dość uniwersalny mechanizm ochrony stosowalny wobec dowolnych aplikacji – ograniczanie środowiska wykonania. Następnie przedstawione zostaną najistotniejsze problemy dotyczące popularnych usług aplikacyjnych – WWW i poczty elektronicznej.

## Bezpieczne środowisko aplikacyjne

Jednym z najważniejszych środków ochrony aplikacji użytkowych przed zagrożeniami płynącymi z zewnątrz i skutkującymi przejęciem kontroli nad aplikacją, a potencjalnie dalej – nad całym systemem operacyjnym, jest stworzenie bezpiecznego środowiska aplikacyjnego, czyli takiego, w którym aplikacja zostaje uruchomiona w specjalnie spreparowanym podsystemie, który minimalizuje zagrożenia.

Ograniczanie środowiska wykonania aplikacji ma na celu w istocie nie tyle uniemożliwienie ataku w ogóle, co minimalizowanie szkód po ewentualnym ataku. Koncepcja działania tego mechanizmu jest następująca:

- zawsze uruchamiamy proces z najmniejszymi wystarczającymi mu uprawnieniami
- ograniczamy przestrzeń aktywności procesu (dozwolonych modyfikacji) do wybranego zawężonego fragmentu systemu, w szcz. systemu plików – tworząc tzw. piaskownicę (ang. sandbox)

W systemach z rodziny Unix popularnym narzędziem służącym do tworzenia piaskownic jest systemowa funkcja chroot(). Jest to uprzywilejowana funkcja systemowa ograniczająca proces do określonego poddrzewa systemu plików. Blokuje jedynie dostęp do plików, tworząc tzw. więzienie. Uwięziony proces nie może otworzyć (w tym utworzyć) pliku poza ograniczonym obszarem, chociaż może dziedziczyć deskryptory wskazujące na pliki spoza tego obszaru.

Tworząc piaskownicę w systemie Unix trzeba stworzyć więzionemu procesowi iluzję pracy w pełnoprawnym systemie. W tym celu w piaskownicy należy zainstalować odpowiednie pliki i katalogi potrzebne programowi i używanym przez niego bibliotekom (na ogół bardzo ograniczone fragmenty /etc, /lib czy /usr/lib).

Mimo ogromnej przydatności funkcji chroot(), związane są z jej wykorzystaniem pewne problemy. Większość z nich dotyczy ataków DoS. I tak przykładowo, mimo ograniczenia środowiska wykonania:

- dysk może się przepełnić (np. zrzutami obrazu pamięci, plikami raportów) – na szczęście w systemie Unix możemy ograniczać programy do oddzielnej partycji
- może nastąpić przepełnienie pamięci – proces może zagarnąć tyle pamięci, że zablokuje to urządzenie z plikiem wymiany – przeważnie możemy ograniczać użycie pamięci
- zużywanie czasu procesora – tu do obrony mamy do dyspozycji polecenie *nice*
- brak automatycznej kontroli nad komunikacją sieciową pozwala potencjalnie wyzwolić się częściowo z uwięzienia

W systemie Unix istnieje polecenie chroot (dostępne z powłoki), które wywołuje funkcję chroot(). Polecenie to ma też pewne ograniczenia, z których najważniejsze to:

- polecenie chroot musi znajdować się w piaskownicy
- chroot wymaga uprawnień superużytkownika
- brak mechanizmu zmiany UID i GID procesu – proces uwięziony wykonuje się z prawami superużytkownika root (ew. sam musi zmienić efektywny UID/GID)
- potencjalne luki umożliwią ucieczkę z piaskownicy (prawa superużytkownika)

W nowszych wydaniach systemów Unix/Linux istnieją inne, doskonalsze narzędzia – np. chrootuid, jail – które przed wywołaniem funkcji chroot() pozwalają zmienić UID oraz GID

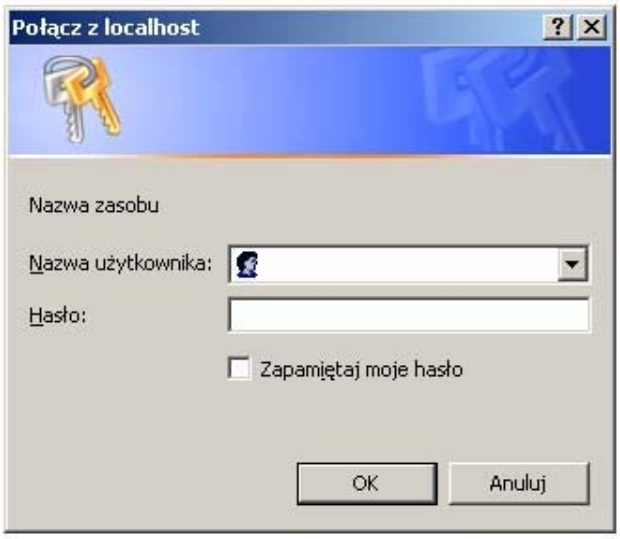
```
jail –u nobody –g www –l /tmp/jail.log –d / /usr/apache /bin/httpd
```

## Usługa WWW

### Uwierzytelnianie

Jednym z ważniejszych problemów zapewnienia podstawowych własności bezpieczeństwa jest, jak wiemy, poprawne uwierzytelnianie. Prosty mechanizm uwierzytelniania został wbudowany w protokół usługi WWW – HTTP. Uwierzytelnianie podstawowe w protokole HTTP przebiega następująco:

- serwer WWW może w dowolnym momencie zażądać od przeglądarki dokonania uwierzytelnienia użytkownika
- przeglądarka wyświetla stosowne okno dialogowe lub podobny obiekt (rysunek 1), który pozwoli użytkownikowi na wprowadzenie danych uwierzytelniających
- po ich pierwszym wpisaniu przeglądarka zapamięta je i automatycznie prześle do serwera na każde następne żądanie
- dane przesyłane są w postaci jawnej
- dane te zostaną usunięte z pamięci z chwilą zamknięcia okna przeglądarki




**Rysunek 1. Okienko uwierzytelniania wyświetlane w przykładowej przeglądarce www**

Wiele implementacji serwerów usługi www umożliwia automatyzację operacji uwierzytelniania, pozwalając na weryfikację otrzymanych danych uwierzytelniających poprzez zewnętrzne bazy danych, przechowujące konfigurację kont użytkowników. Przykładowo, serwer Apache posiada rodzinę modułów *mod_auth* służacych do tego celu (np. *mod_auth_mysql*), a serwer IIS pozwala na starowanie automatyzają poprzez ustawienia opcji *Panel Sterowania* → *Narzędzia Administracyjne* → *Menedżer Usług Internetowych* (można zdefiniować np. uwierzytelnianie użytkownika przez domenę)

W standardzie HTTP 1.1 uwzględniono uwierzytelnianie kryptograficzne, realizowane najczęściej z wykorzystaniem algorytmu MD5. Niestety nie przewidziano w specyfikacji dwustronnego uwierzytelniania.

Rozwiązaniem problemów uwierzytelniania, które przyjęło się w praktyce jest zastosowanie niezależnie od protokołu aplikacyjnego HTTP, protokołu sesji – SSL.

### Protokół SSL (*Secure Sockets Layer*)

W istocie protokół SSL tworzy tunel kryptograficzny usługi www. Tak zabezpieczona usługa znana jest pod nazwą https (port 443/tcp). Jednym z podstawowych komponentów protokołu SSL jest protokół uzgadniania, który realizuje zadania uwierzytelniania stron.

Protokół uzgadniania (*handshake protocol*) działa wg poniższego schematu:

- klient wysyła do serwera komunikat `ClientHello` (wersja protokołu, identyfikator sesji, listę obsługiwanych szyfrów i metod kompresji, dane losowe)
- serwer odsyła komunikat `ServerHello` (wersja protokołu, identyfikator sesji, wybrany szyfr i metodę kompresji, dane losowe oraz swój certyfikat X.509) oraz opcjonalnie – żądanie certyfikatu klienta (wraz z losowym zawołaniem)
- klient uwierzytelnia serwer na podstawie odebranego certyfikatu i w razie niepowodzenia przerywa połączenie

- po pomyślnym uwierzytelnieniu klient tworzy pierwotny sekret główny (*premaster secret*), który szyfruje kluczem publicznym serwera i wysyła do serwera
- jeśli serwer żądał uwierzytelnienia klienta, to klient wysyła też swój certyfikat oraz podpisane zawołanie odebrane wcześniej od serwera
- po ewentualnym uwierzytelnieniu klienta serwer deszyfruje pierwotny sekret główny i na jego podstawie uzyskuje sekret główny (*master secret*), podobnie czyni w tym czasie klient
- z wygenerowanego sekretu głównego obie strony tworzą (zależny od ustalonego algorytmu szyfrującego) klucz sesji (lub klucze sesji – do szyfrowania i podpisywania)
- klient i serwer wysyłają do siebie nawzajem zaszyfrowany kluczem sesji komunikat o zakończeniu fazy uzgadniania
- protokół uzgadniania kończy się i (o ile wzajemna weryfikacja przebiegła pomyślnie) rozpoczyna się sesja SSL

Poprawność procedury uwierzytelniania wynika z następujących obserwacji:

- jeśli serwer nie posiadałby klucza prywatnego odpowiadającego kluczowi publicznemu ze swojego certyfikatu:
  - nie rozszyfruje poprawnie sekretu i nie wygeneruje tego samego klucza sesji co klient
  - wówczas połączenie zastanie przerwane w fazie uzgadniania
  - stąd klient ma pewność, że serwer jest tym, czyją autentyczność poświadcza certyfikat (po weryfikacji jego poprawności)
- jeśli klient nie posiadałby klucza prywatnego odpowiadającego kluczowi publicznemu ze swojego certyfikatu:
  - serwer pobierze jego klucz publiczny z certyfikatu i rozszyfruje podpisane kluczem prywatnym klienta zapytanie
  - nie otrzyma tego, które sam wysłał
  - zatem klient nie jest tym, czyją autentyczność poświadcza certyfikat

Newralgiczna w tym procesie jest weryfikacja certyfikatów – SSL jest podatny na atak *man-in-the-middle*. Sposobem redukcji zagrożenia może być np. weryfikacja czy adres IP asocjacji z nawiązanego połączenia zgadza się z adresem IP w certyfikacie.

## Luki bezpieczeństwa w usłudze WWW

Luki bezpieczeństwa w usłudze WWW dotyczą wielu komponentów systemu, w szczególności klientów (przeglądarek), serwerów czy środowiska wykonania (systemu operacyjnego).

### Przeglądarki WWW

Wśród typowych problemów bezpieczeństwa klientów usługi WWW należy wymienić chociażby

- problemy z losową generacją kluczy SSL
- błędy implementacji S/MIME (*Secure/Multi-purpose Internet Mail Extension*)

- problemy specyficzne dla konkretnych przglądarek, np. Internet Explorer
  - BHO (*Browser Helper Objects*) – pozwala na integrację z przeglądarką np. toolbarów (praktycznie dowolnych aplikacji) – często wykorzystywane przez malware do cichej instalacji (zagrożenia browser hijacking – podmiana parametrów pracy przeglądarki, takich jak adres strony domowej, tracking – śledzenie pracy użytkownika, niechciane wyskakujące okienka pop-ups)
- konie trojańskie / wirusy w dokumentach hipertekstowych – ochronę tu stanowić może wiele mechanizmów
  - zamknięte środowisko uruchomieniowe (*sandbox*)
  - korzystanie wyłącznie z poprawnie zdefiniowanego zbioru zaufanych serwerów źródłowych
  - certyfikaty cyfrowe

Z ostatnim z powyższych problemów związanych jest szereg zagrożeń niesionych przez języki automatyzacji operacji na dokumentach hipertekstowych i definicji dynamicznych stron DHTML czy .NET. Języki te to przede wszystkim Java i ActiveX

Java charakteryzuje się w tym kontekście następującymi własnościami:

- jest to język stosunkowo bezpieczny (brak wskaźników i problemu przepełnienia bufora)
- jest językiem interpretowanym (aplety mają format *byte code*) – możliwe są luki bezpieczeństwa w programie interpretera
- maszyna wirtualna JVM posiada wbudowany system ochronny: analizatory kodu (code verifier i class loader), sandbox, security manager, certyfikacja serwerów

ActiveX posiada następujące istotne w naszych rozważaniach cechy:

- program jest dystrybuowany w postaci skompilowanej – praktycznie brak tu możliwości analizy bezpieczeństwa
- system ochronny: certyfikacja kontrolek ActiveX (mechanizm *authenticode*) – praktyka pokazuje, iż generalnie zaufanie jedynie certyfikacji jest złudne.

W praktyce, najczęściej wbudowane w usługę WWW i dostępne w przeglądarkach mechanizmy ochrony są uzupełniane o filtrację treści ruchu HTTP na zaporze firewall (osobistej lub sieciowej).

**Serwery WWW**

Do charakterystycznych problemów bezpieczeństwa, na które cierpią implementacje serwerów usługi WWW można zaliczyć w szczególności np.:

- tylne furtki
- błędy przepełnienia bufora, np. w ism.dll (uruchamianym przez IIS dla URL *.htr) pozwala na uruchomienie kodu w kontekście procesu serwera z jego uprawnieniami

### Środowisko wykonania

Problemy środowiska wykonania szczególnie często ujawniają się w przypadku systemu MS Windows. Tu wymienimy chociażby:

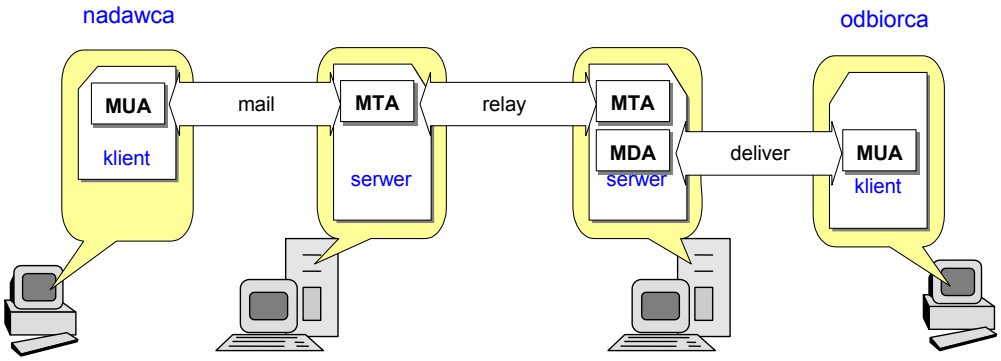
- mechanizm LSP (*Layered Service Provider*) oferujący możliwość podpinania się modułów LSP (praktycznie dowolnych aplikacji, w tym malware) pod stos protokołów w bibliotece Winsock 2
- luki w implementacjach protokołów MHTML, MS-ITS, XMLHTTP i VBScript (wykorzystywane przez Outlook, WindowsUpdate oraz rozliczne malware)

Umożliwiają one instalację niechcianego oprogramowania pobieranego nieświadomie poprzez usługę WWW.

## Poczta elektroniczna

Rysunek 2 przedstawia model komunikacji systemu internetowej usługi pocztowej standardu SMTP (RFC 821). Wyróżnione na nim elementy systemu to:

- **MUA** = Mail User Agent
- **MTA** = Mail Transfer Agent
- **MDA** = Mail Delivery Agent




**Rysunek 2. Model komunikacji SMTP**

Podstawowe problemy bezpieczeństwa dotyczące poczty obejmują:

- niepożądane przesyłki (*spam*)
- niebezpieczne załączniki (wirusy)
- potwierdzanie dostarczenia
- naruszenie poufności / integralności / autentyczności

## Spam

Pojęcie spam dotyczy ogółu niechcianych przesyłek pocztowych zajmujących zasoby pamięciowe (skrzynka pocztowa odbiorcy) i czasowe systemu. Ochrona anty-spamowa sprowadza się do odfiltrowania takich przesyłek z całości ruchu pocztowego i może być realizowana na kilku poziomach modelu komunikacji SMTP.

Na poziomie MTA filtracja jest dokonywana poprzez analizę nagłówka SMTP, np.

- adresów: czy są weryfikowalne w DNS, czy odpowiadają rekordom MX czy nie jest na czarnej liście
- weryfikacja konta nadawcy (komenda VRFY protokołu SMTP)

Posiada ona istotną zaletę – oszczędność zasobów – odrzucamy spam na pierwszej linii obrony. Wadą tego rozwiązania jest mała precyzja wynikająca z faktu, iż na tak wczesnym etapie posiadamy mało informacji do dyspozycji. Stąd występuje duże prawdopodobieństwo pomyłki – sklasyfikowania niechcianej przesyłki jako pożądanej i odwrotnie.

Stosunkowo dużą skuteczność i małe efekty uboczne (opóźnienie) wykazuje tu mechanizm znany jako szare listy (*greylisting*). Mianowicie, po odebraniu przesyłki MTA odsyła na adres nadawcy kod 452 „czasowa niedostępność" i czeka na powtórną transmisję. Automaty spamerskie z założenia nie retransmitują, stąd akceptowane jako pożądane są wszystkie retransmitowane listy.

Poziom MDA pozwala stosować do realizacji możliwie skutecznej filtracji takie rozwiązania jak:

- analiza heurystyczna (na podstawie przygotowanej bazy danych charakterystycznych)
- analiza statystyczna (samouczące się filtry Bayesa)

Również klienci pocztowi MUA posiadają często wbudowane narzędzia filtrujące, które, niekiedy automatycznie, pozwalają użytkownikowi klasyfikować wybrane listy jako spam.

### Ochrona kryptograficzna poczty

Powszechnie spotykane są następujące standardy ochrony kryptograficznej

- PEM – *Privacy Enhanced Mail*
- PGP – *Pretty Good Privacy*
- S/MIME – *Secure* MIME

Istnieje wiele kompleksowych systemów i standardów pocztowych wykorzystujących kryptografię, w tym przykładowo:

- X.400 MHS – Message Handling System
- EDI (EDIFACT – X.435) – Electronic Data Interchange

### System PEM (*Privacy Enhanced Mail*)

PEM to jeden z pierwszych standardów zaproponowanych do ochrony przesyłek protokołu SMTP i posiadający zgodny z pierwotnymi wymaganiami tego protokołu format RFC822 (rysunek 3). W PEM możliwa jest przede wszystkim kontrola integralności przy wykorzystaniu

MD2 lub MD5 (128b) – zgodnie ze standardem RFC1421. Opcjonalnie możliwe jest szyfrowanie wiadomości – DES-ECB, 3DES (RFC1423). PEM wspiera zarządzanie kluczami i certyfikację wg ISO X.509 (RFC1422, RFC1424).


**Rysunek 3. Schematyczna struktura przesyłki pocztowej zabezpieczonej kryptograficznie**

Przykładowe implementacje to chociażby RIPEM czy TIS-PEM. Jednak w szerszej skali PEM nie uzyskał dużej popularności.

### PGP (*Pretty Good Privacy*)

PGP powstał jako projekt akademicki prowadzony przez Phila Zimmermanna (z MIT). Prace uwieńczyły standard IETF RFC1991 (PGP 2.6.x) i wielka popularność jaką zyskał on w Internecie. W 1998 IETF zatwierdził standard OpenPGP (RFC2440) opracowany przez Network Associates, bazujący na PGP 5.x. Istnieje również alternatywny projekt PGPi (www.pgpi.org) = International PGP – przeniesiony poza USA. Z najpopularniejszych implementacji wymienić należy Desktop PGP – jest to komercyjna wersja rozprowadzana przez Network Associates (rozszerzona np. o IDS) – oraz GnuPG – wersję dystrybuowaną na licencji GNU. Ponadto wiele aplikacji wykorzystuje PGP (np. enigmail – rozszerzenie klientów pocztowych Mozilli).

PGP umożliwia:

- szyfrowanie wiadomości pocztowej
  - wykorzystywany jest jednorazowy klucz symetryczny generowany dla każdego szyfrowanego listu
  - następnie klucz ten szyfrowany jest metodą asymetryczną – kluczem publicznym odbiorcy – Diffie-Hellman/DSS, RSA (768b, 1024b, ...)
  - i tak zaszyfrowany klucz jest dołączany do zaszyfrowanego listu
- kontrolę integralności – MD5, SHA-1
- symetryczne szyfrowanie dowolnych plików

**S/MIME**

Standard Secure MIME umożliwia wygodną integrację mechanizmu kryptograficznego zabezpieczenia korespondencji pocztowej z protokołem SMTP, poprzez wykorzystanie rozszerzenia uznanego mechanizmu obsługi załączników MIME. Wersja S/MIME 1 została opracowana przez RSA Security w 1995r. i wykorzystywała mechanizmy kryptograficzne wchodzące w skład PKCS (*Public Key Cryptography Standards*). Wersja S/MIME 2 (RFC2311/12) powstała w 1998r., a wkrótce później S/MIME 3 (RFC 2630-34). S/MIME 3 oferuje m.in. rozszerzone funkcje bazujące na mechanizmach MSP (*Message Security Protocol* protokołu opracowanego pierwotnie dla *Defense Message System*).

Należy nadmienić istnienie również protokołów PGP/MIME i OpenPGP/MIME.

## Pytania problemowe

1. W przykładzie statycznych reguł filtracji z rysunku 5 zdefiniowano 4 reguły. Jedna z nich jest jednak nadmiarowa i można ją usunąć bez żadnych konsekwencji dla przebiegu filtracji. Która to reguła?