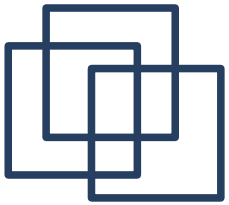
# Bezpieczeństwo

1. Zagrożenia w sieci  WWW.

• przykłady ataków na serwisy internetowe.

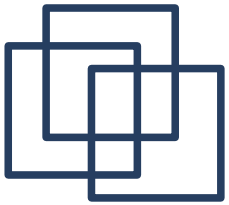
2. Bezpieczeństwo przesyłanych informacji przez sieć TCP

- protokół SSL,
- protokół S-HTTP,
- protokoły SSH.



# Przykłady ataków

Istnieje wiele możliwości ataków na serwis internetowy. Począwszy od prób podsłuchu, poprzez włamania wykorzystujące luki w systemie operacyjnym i oprogramowaniu serwera, kończąc na preparowaniu odpowiednich danych przesyłanych przez protokół HTTP w celu oszukania skryptów generujących dynamiczną zawartość stron.
W dalszej części zaprezentowano podstawowe techniki z ostatniej grupy.



# Przykłady ataków

**Cross-site request forgery (CSRF)**:

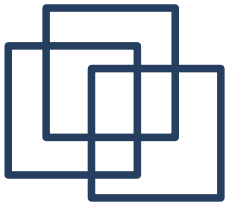
Błąd dotyczy np. forów dyskusyjnych z możliwością umieszczania plików graficznych. Pozwala to na wykonanie dowolnego żądania HTTP.

Przykład: BBCode

```
[img]http://www.jakies-forum.pl/admin/delete_user.php?uid=1[/img]
```

Jeśli administrator/moderator uruchomi stronę z takim „obrazkiem” to skasuje konto użytkownika.

**Ochrona**: istotne funkcje dostępne wyłącznie przez POST.



# Przykłady ataków

**Session fixation**:

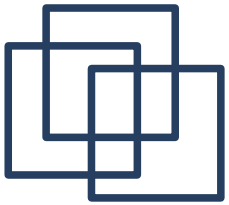
Przejęcie kontroli nad kontem (zalogowanego) użytkownika.

Użytkownikowi należy wysłać link, który zostanie przez niego użyty:

`http://www.jakis-adres.pl?SID=123456`

Użytkownik po zalogowaniu będzie identyfikowany przez przekazany mu numer sesji.

**Ochrona:** regeneracja identyfikatora sesji, powiązanie sesji z IP.



# Przykłady ataków

**Iniekcje (SQL Injection, Code Injection, ...)**:

Używamy adresu:

`http://jakis-serwis.pl/login.php?id=1%20OR%201=1`

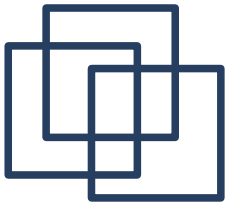
co odpowiada `( $_GET['id'] = "1 OR 1=1" )`

Jeśli w kodzie kod autoryzujący jest realizowany np. poprzez zapytanie

```
mysql_query('SELECT name FROM admins WHERE id = '.$_GET['id']);
```

to zyskamy prawa administratora

**Ochrona:** analiza wprowadzanych wartości.



# Przykłady ataków

**Directory traversal**:

Atak typu **Cookie injection**. Umożliwia przeglądanie systemu plików poprzez wysłanie odpowiedniego cookie.

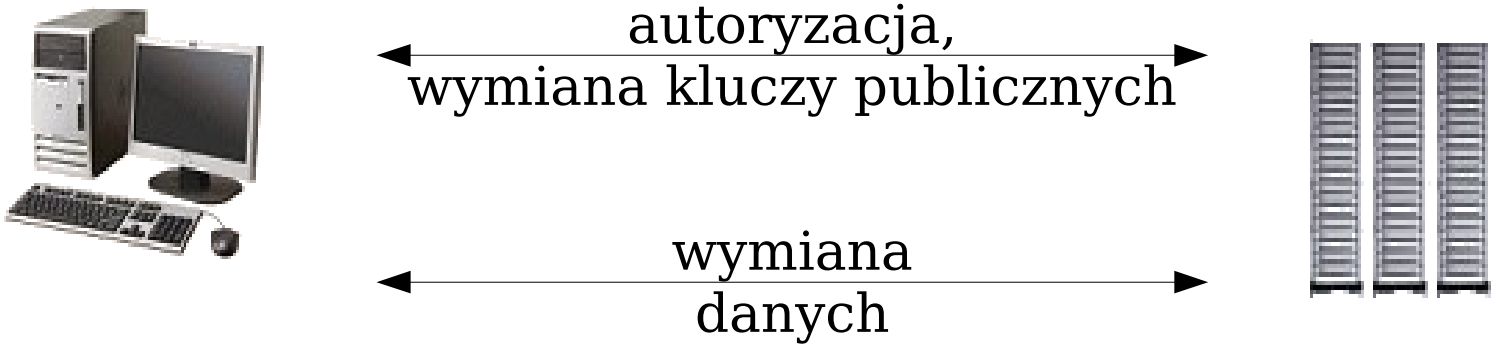
```
// GET /index.php HTTP/1.0
// Cookie: skin=../../../../../../../../etc/passwd

$skin = isset($_COOKIE['skin']) ? $_COOKIE['skin'] :
                                                'default.skin';
readfile('gfx/skins/'.$skin);
```

**Ochrona**: analiza zawartości cookies.

Inne: http://anakin.us/blog/php-bezpieczne-programowanie/.

# Prywatność i bezpieczeństwo

1. **Protokół SSL/TLS** *(Secure Socket Layer / Transport Layer Security)* – służy do wymiany danych, spełnia trzy warunki ochrony informacji: integralność, uwierzytelnianie oraz poufność. Kodowaniem objęte są wszystkie dane wysyłane w ramach połączenia. SSL został opracowany w 1994 roku.
2. **Protokół S-HTTP** *(Secure HTTP)* – stanowi rozszerzenie protokołu HTTP. Służy do bezpiecznego przesyłania pojedynczych danych. Opracowany w 1995 roku.



# Podstawy kryptografii

1. Kody podstawieniowe.
2. Kodowanie z wykorzystaniem klucza.

- klucz symetryczny – do zakodowania i rozkodowania informacji potrzebny jest ten sam klucz. Potrzebny jest bezpieczny “kanał” w celu wymiany klucza pomiędzy nadawcą i odbiorcą.
- klucz asymetryczny – do zakodowania używa się innego klucza niż do rozkodowania. Jeden z nich to klucz prywatny, drugi – klucz publiczny. Znajomość klucza publicznego **nie ułatwia** znalezienia klucza prywatnego.

# Podstawy kryptografii

Schematy działania:

- klucz symetryczny

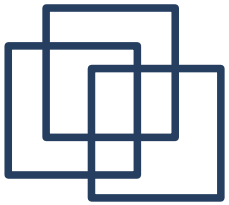
wiadomość $\xrightarrow{A(K)}$ wiadomość zakodowana $\xrightarrow{A^{-1}(K)}$ wiadomość

- klucz asymetryczny

wiadomość $\xrightarrow{A(Publ)}$ wiadomość zakodowana $\xrightarrow{B(Pryw)}$ wiadomość

- klucz asymetryczny umożliwia podpisywanie wiadomości:

wiadomość $\xrightarrow{C(Pryw)}$ wiadomość zakodowana $\xrightarrow{D(Publ)}$ wiadomość



# Protokół SSL

1. **Protokół SSL** działa ponad warstwą transportową (TCP). Dzięki temu może być wykorzystywany przez wszystkie protokoły warstw wyższych. Jego główne cele:

- **poufność** – dane są przekazywane w **bezpiecznej** postaci zaszyfrowanej,
- **wszechstronność** – niezależni programiści mogą tworzyć programy mogące się wzajemnie komunikować i uzgadniać parametry transmisji,
- **rozszerzalność** – nowe techniki kryptograficzne mogą być łatwo implementowane w ramach istniejącej technologii,
- **efektywność** – wykorzystanie zasobów takich jak czas procesora, liczba połączeń sieciowych czy przepustowość sieci powinno być optymalne.

# Protokół SSL

**Nawiązanie połączenia** *(SSL Handshake Protocol)*.

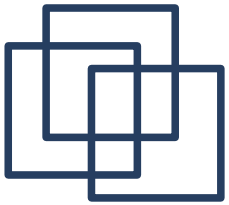
Client Hello →

← Server Hello
Certyfikat
Klucz publiczny
Prośba o certyfikat
Server Hello Done

Certyfikat
Klucz publiczny
Sprawdzenie certyfikatu serwera
[Zmiana algorytmu szyfrującego]
Finished →

← [Zmiana algorytmu szyfrującego]
Finished

W przypadku odnowienia sesji szare komunikaty nie są przesyłane.



# Protokół SSL

Client Hello – przykład:

```
80 67 01 03 01 00 4e 00 00 00 10 01 00 80 03 00
80 07 00 c0 06 00 40 02 00 80 04 00 80 00 00 39
00 00 38 00 00 35 00 00 33 00 00 32 00 00 04 00
00 05 00 00 2f 00 00 16 00 00 13 00 fe ff 00 00
0a 00 00 15 00 00 12 00 fe fe 00 00 09 00 00 64
00 00 62 00 00 03 00 00 06 ca ce 92 2d 83 bd 80
8f 4a 34 06 aa 23 78 fb 44
```

kolejno: długość (103), kod komunikatu “Client Hello”, wersja, długość bloku obsługiwanych algorytmów szyfrujących (78), identyfikator sesji, długość bloku identyfikatora (16), obsługiwane algorytmy szyfrujące (3 bajty – jeden algorytm), identyfikator.

# Certyfikat SSL

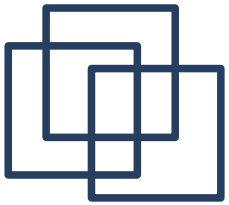
**Certyfikat** pozwala potwierdzić autentyczność strony WWW. Aby certyfikat był ważny musi być podpisany, przez “zaufaną” osobę lub organizację.

Podpisywanie wiadomości (certyfikatu):

- wygenerowanie skrótu wiadomości (MD5),
- zakodować go kluczem prywatnym,

Wiadomość jest rozprowadzana wraz z zakodowanym skrótem i (znanym powszechnie) kluczem publicznym podpisującego. W celu sprawdzenia autentyczności należy:

- rozkodować skrót kluczem publicznym podpisującego
- porównanie go ze skrótem wyliczonym samodzielnie.



# Biblioteka OpenSSL

**OpenSSL** [http://www.openssl.org]

funkcje wykorzystywane do nawiązania połączenia **`openssl/ssl.h`**

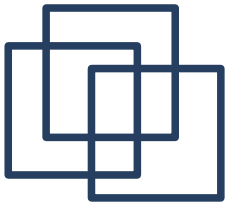
```
int SSL_library_init(void);
```
// inicjalizacja biblioteki

```
SSL_CTX *SSL_CTX_new(SSL_METHOD *method);
```
// przygotowanie obiektu obsługującego połączenie, jako argument można podać np. **`SSL_METHOD SSLv23_method(void).`**

```
SSL *SSL_new(SSL_CTX *ctx);
```
// obiekt zarządzający transmisją SSL.

```
int SSL_set_fd(SSL *ssl, int fd);
```
// powiązanie z deskryptorem

```
int SSL_connect(SSL *ssl);
```
// nawiązanie połączenia SSL.



# Biblioteka OpenSSL

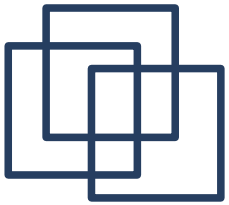
```
int SSL_read(SSL *ssl, void *buf, int num);    // czytanie
```
z SSL.

```
int SSL_write(SSL *ssl, const void *buf, int num);  //
```
pisanie z SSL.

```
int SSL_shutdown(SSL *ssl); // zamknięcie połączenia SSL.
```

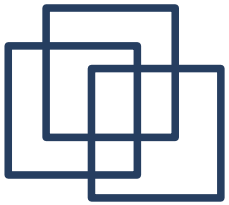
Dodatkowo biblioteka udostępnia wiele funkcji przydatnych w kryptografii:

- algorytmy obliczające sumy kontrolne np. SHA, MD5, HMAC
- algorytmy szyfrujące np. AES, DES, RSA,
- generowanie dużych liczb losowych, generowanie kluczy.



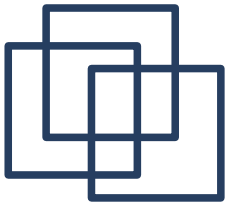
# SSL w Javie

```
import java.io.*;
import javax.net.ssl.*;

public class SSLSocketClient {

    public static void main(String[] args) throws Exception {
        try {
            SSLSocketFactory factory =
                (SSLSocketFactory)SSLSocketFactory.getDefault();
            SSLSocket socket =
                (SSLSocket)factory.createSocket
                                        ("www.verisign.com", 443);
            socket.startHandshake();
            PrintWriter out = new PrintWriter(
                new BufferedWriter(
                    new OutputStreamWriter(
                        socket.getOutputStream())));
```



# SSL w Javie

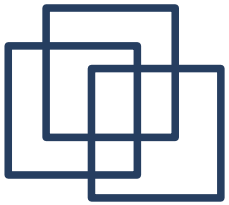
```
            out.println("GET / HTTP/1.0");
            out.println();
            out.flush();

            BufferedReader in = new BufferedReader(
                new InputStreamReader(socket.getInputStream()));

            String inputLine;
            while ((inputLine = in.readLine()) != null)
                System.out.println(inputLine);

            in.close();
            out.close();
            socket.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```



# Protokół S-HTTP

Podstawowe własności S-HTTP:

- nowa komenda **`Secure * Secure-HTTP/1.2`**. W odpowiedzi serwer przesyła informację o wykorzystywanych technologiach szyfrowania,
- używa szyfrów symetrycznych. Wymiana tajnych kluczy następuje z wykorzystaniem szyfrów asymetrycznych lub innych metod autoryzacji (np. Kerberos),
- nie wprowadza dodatkowych protokołów pośredniczących w transmisji danych – działa na wyższym poziomie niż SSL.



# Uzgadnianie klucza symetrycznego

Protokół Diffiego-Hellmana:

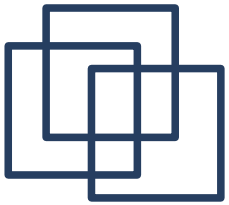
Dwie strony komunikacji **A** i **B** używające prywatnych kluczy **SK(A)** i **SK(B)**.

Klucze publiczne obliczamy ze wzorów:

$$PK(A) = L^{SK(A)} \bmod M, \qquad PK(B) = L^{SK(B)} \bmod M.$$

gdzie **M** to liczba pierwsza, a **L** liczba naturalna mniejsza niż **M**.

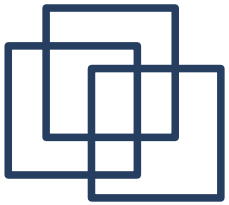
Klucz wspólny: $CK(A, B) = PK(A)^{SK(B)} \bmod M = PK(B)^{SK(A)} \bmod M$.



# Protokół SSH

**SSH** *(Secure Shell)* jest następcą usługi Telnet. Umożliwia on wykonywanie poleceń na zdalnym komputerze. Przesyłane informacje są kodowane. Usługa jest dostępna przez port 22. Protokół SSH dzieli się na kilka warstw:

- protokół warstwy transportowej *(Transport Layer Protocol),*
- protokół autoryzacji *(User Authentication Protocol),*
- protokół połączenia *(Connection Protocol),*
- protokoły użytkownika *(Client Protocols)*.



# SSH – protokół warstwy transportowej

[serwer] `SSH-protokół-oprogramowanie komentarz`

[klient] `SSH-protokół-oprogramowanie komentarz<CRLF>`

przykład: `SSH-2.0-OpenSSH_3.6.1p2`

[serwer] `<algorytmy obsługiwane przez serwer>`

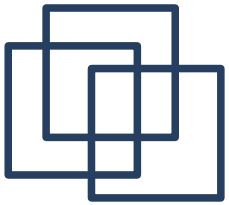
[klient] `<algorytmy używane przez klienta>`

[klient-serwer] `<wymiana kluczy publicznych`

[klient-serwer] Po uzgodnieniu algorytmów transmisja jest kodowana.

Cechy SSH (w porównaniu do SSL)

- różne algorytmy używane do transmisji w każdą ze stron,
- zwiększona uniwersalność kosztem bezpieczeństwa.

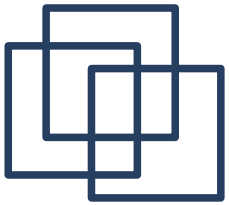


# Protokół SSH

Najważniejsze zastosowania SSH

- ssh – następca telnet, rlogin,
- scp (Secure Copy) – narzędzie do kopiowania plików, następca rcp,
- sftp (Secure FTP) – protokół do transferu plików, następca FTP.

Oprogramowanie OpenSSH (http://www.openssh.com) udostępnia implementacje klientów i serwerów wspomnianych usług.

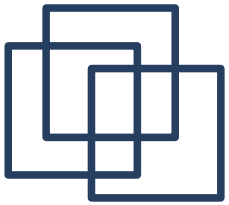


# Firewall - iptables

Aktualna konfiguracja: **`iptables-save > konfiguracja.txt`**:

```
# Generated by iptables-save v1.3.5 on Tue Oct 24 12:36:50 2006
*filter
:INPUT DROP [5203:1000594]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [158054:29139188]
-A INPUT -i lo -j ACCEPT
-A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
-A INPUT -s 192.168.0.0/255.255.255.192 -i eth0 -p tcp -m state
  --state NEW -m tcp --dport 22 -j ACCEPT
-A INPUT -s 143.232.11.26 -i eth0 -p tcp -m state --state NEW -m
  tcp --dport 22 -j ACCEPT
-A INPUT -p tcp -m state --state NEW -m tcp --dport 80 -j ACCEPT
-A INPUT -p tcp -m state --state NEW -m tcp --dport 25 -j ACCEPT
-A INPUT -p tcp -m state --state NEW -m tcp --dport 995 -j ACCEPT
-A INPUT -p icmp -m icmp --icmp-type 8 -j ACCEPT
COMMIT
# Completed on Tue Oct 24 12:36:50 2006
```

Ustawianie konfiguracji: **`iptables-restore konfiguracja.txt`**



# Podsumowanie

Bezpieczeństwo w sieciach jest najczęściej zapewniane przez algorytmy z kluczem asymetrycznym. Dzięki nim, możemy nie tylko skutecznie szyfrować przesyłaną treść, ale również zidentyfikować drugą stronę komunikacji. Najczęściej wykorzystuje się tutaj rozwiązania oparte o protokół SSL.