*Michał Leszczyński*

# Certyfikaty użytkownika SSL – jak to ugryźć?

Technologia SSL znajduje swoje użycie w olbrzymiej ilości aplikacji i usług internetowych. Jednym z głównych celów jej zastosowania jest umożliwienie klientowi sprawdzenie, czy komunikuje się on z zaufanym serwerem. Istnieje jednak rozszerzenie, które pozwala dokonać zupełnie odwrotnej czynności, umożliwiając serwerowi sprawdzenie autentyczności klienta. Celem tego artykułu jest zatem omówienie i implementacja certyfikatów użytkownika na przykładzie aplikacji hostowanej w kontenerze Tomcat.

Najpopularniejszą metodą logowania w Internecie jest uwierzytelnianie za pomocą hasła. Metoda ta, choć jest jedną z najwygodniejszych, nie jest pozbawiona wad. W celach porównawczych na serwerze muszą być przechowywane hasła wszystkich użytkowników – najczęściej w formie hasha, np. z funkcji *SHA1*, *bcrypt* etc. W przypadku wycieku bazy danych będącego choćby następstwem włamania, atakujący może posłużyć się różnymi technikami, aby odzyskać oryginalne hasła, inwestując w to odpowiednią ilość mocy obliczeniowej. Istnieje też ryzyko wstawienia przez włamywacza backdoora w kodzie aplikacji, który zajmie się rejestrowaniem haseł przesyłanych na serwer otwartym tekstem podczas logowania (ruch HTTPS trafiający do aplikacji zostaje już wcześniej odszyfrowany przez serwer).

Odporność na tego typu zagrożenia wykazuje autoryzacja przy pomocy certyfikatu użytkownika SSL, podczas której na serwer nigdy nie jest wysyłany pełny certyfikat, a jedynie dowód na to, że użytkownik, który go prezentuje, jest równocześnie jego właścicielem. Jest to możliwe dzięki zastosowaniu kryptografii asymetrycznej. Po stronie serwera nie trzeba więc przechowywać żadnych wrażliwych danych logowania dotyczących konkretnych użytkowników. Bezpieczeństwo całego systemu przekłada się bezpośrednio na bezpieczeństwo klucza używanego do wystawiania certyfikatów.

### Podpis cyfrowy

Algorytmy szyfrowania asymetrycznego, oprócz swojego typowego zastosowania w szyfrowaniu wiadomości, znajdują użycie przy tworzeniu podpisów cyfrowych. Para kluczy dla szyfru asymetrycznego składa się z dwóch kluczy: prywatnego (który należy do właściciela i powinien być przechowywany bezpiecznie) oraz publicznego. Dowolna wiadomość może zostać podpisana cyfrowo przez posiadacza klucza prywatnego, a wiarygodność podpisu można zweryfikować, wykorzystując klucz publiczny.

*Rysunek 1. Procedura wystawiania certyfikatów użytkownika SSL*

### Distinguished name

Certyfikaty użytkownika są zgodne ze standardem X.509 (RFC 5280[1]), muszą więc zawierać *distinguished name* (DN) podmiotu oraz wystawcy, czyli informacje o nich zapisane w formacie par **`atrybut=wartość`**, oddzielanych przecinkiem.

Przykładowy DN podmiotu:

```
CN=someguy123, O=Example Flail Store, OU=Member
```

Najczęściej używane standardowe atrybuty:

| | |
|---|---|
| O | organization name |
| OU | organizational unit name |
| CN | common name |
| L | locality name |
| ST | state or province |
| C | country name |

## SPOSÓB DZIAŁANIA

### Wystawianie certyfikatów

Certyfikaty SSL użytkownika mogą być wydawane przez określony urząd certyfikacji (CA), który posiada swój certyfikat główny i jest uznany za zaufany na serwerze aplikacji. Ponieważ certyfikaty użytkownika są wystawiane w celu umożliwienia serwerowi sprawdzenia tożsamości klienta, a nie odwrotnie jak w przypadku klasycznego zastosowania SSL, certyfikat główny nie musi być zainstalowany w systemie klienta.

### Uwierzytelnianie użytkownika

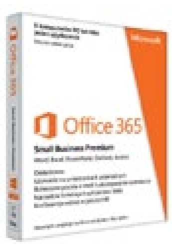
Autoryzacja użytkownika za pomocą certyfikatu jest opcjonalną częścią *SSL handshake*, czyli negocjacji bezpiecznego połączenia z serwerem. Należy tu zauważyć, że przeprowadzenie takiej autoryzacji nie jest możliwe w połączeniu nieszyfrowanym.

Podczas uwierzytelniania certyfikatem użytkownika, losowe dane wygenerowane podczas negocjacji bezpiecznego połączenia są podpisywane przez użytkownika za pomocą wcześniej wystawionego mu certyfikatu. Podpis ten jest później wysyłany wraz z informacjami o certyfikacie do serwera.

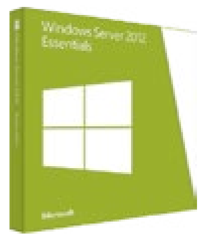
# TTS Company

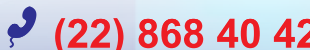
**Największy wybór profesjonalnego oprogramowania w Polsce !**

**… w ofercie programy ponad 300 producentów …**



**www.OprogramowanieKomputerowe.pl**

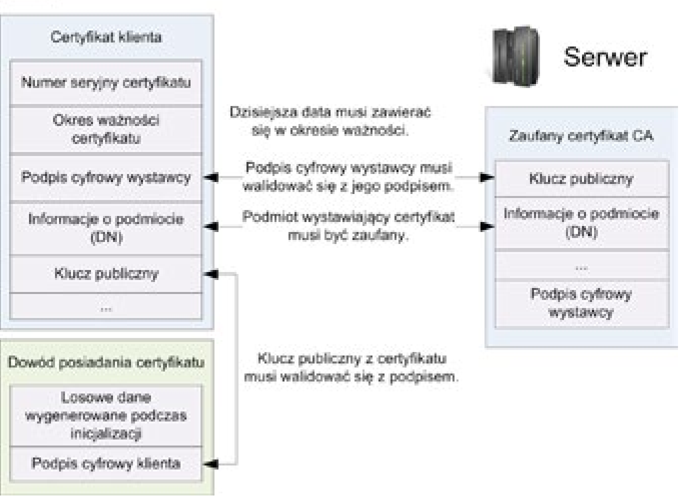


**Więcej informacji:** 

**sales@tts.com.pl**

**Sprzedaż** **Dystrybucja** **Import na zamówienie**

TTS Company Sp. z o.o. ul.Kryniczna 2 03-934 Warszawa e-mail: tts@tts.com.pl www.tts.com.pl




*Rysunek 2. Wymagania, które muszą zostać spełnione przy autoryzacji*

# IMPLEMENTACJA

## Wymagania

Do pracy potrzebne będą:

- serwer Apache Tomcat ze skonfigurowanym connectorem HTTPS,
- zestaw narzędzi **OpenSSL** oraz **keytool**,
- podstawowa wiedza na temat tworzenia servletów [2].

## Konfiguracja kontenera Tomcat

**Generowanie certyfikatu CA**

Na początku konieczne będzie wygenerowanie certyfikatu głównego dla urzędu certyfikacji, wykorzystywanego przez serwer aplikacji do wystawiania i weryfikowania certyfikatów użytkowników. Do utworzenia certyfikatu potrzebny będzie zestaw narzędzi OpenSSL.

Tworzenie nowego klucza prywatnego:

```
$ openssl genrsa – out cakey.pem 2048
```

… oraz certyfikatu CA dla tego klucza:

```
$ openssl req – new – x509 – days 3650 – key cakey.pem –
extensions v3_ca – out cacert.pem
```

Po wywołaniu polecenia użytkownik zostaje zapytany o dane potrzebne do wystawienia certyfikatu. Dane te będą później widniały w każdym certyfikacie klienta jako informacje o wystawcy.

**Import certyfikatu do truststore**

Po utworzeniu certyfikatu CA trzeba będzie go zaimportować do truststore (magazynu zaufanych certyfikatów). Można to zrobić, używając `keytool`:

```
$ keytool – importcert – keystore truststore.jks – alias our_ca
– file cacert.pem
```

Jeżeli truststore nie jest jeszcze utworzony, polecenie utworzy magazyn o nazwie `truststore.jks` i poprosi o ustawienie dla niego hasła.

**Konfiguracja connectora**

Ostatnim krokiem konfiguracji kontenera jest włączenie autoryzacji klientów w connectorze SSL oraz ustawienie opcji związanych z truststore. Wszystkie certyfikaty wystawione przez CA, którego certyfikat znajduje się w zaufanym magazynie, automatycznie będą zaufane.

Do tagu `Connector` odnoszącego się do connectora HTTPS należy dodać następujące atrybuty:

```
truststoreFile="<sciezka do cacerts.jks>"
truststorePass="<haslo magazynu>"
clientAuth="want|true"
```

Atrybut `clientAuth` określa zachowanie serwera podczas negocjacji bezpiecznego połączenia. Jeśli wartość atrybutu zostanie ustawiona na `want`, serwer będzie honorował certyfikaty użytkownika, ale pozwoli również ustanowić połączenie klientom, którzy z jakiegoś powodu nie mogą przedstawić prawidłowego certyfikatu. W przypadku ustawienia wartości na `true`, serwer każdorazowo będzie zrywał połączenia od klientów, którzy się nie uwierzytelnią.

# CLIENT-SIDE: GENEROWANIE ŻĄDAŃ CERTYFIKACJI Z PRZEGLĄDARKI

Do bezpiecznego przeprowadzenia certyfikacji potrzebne jest API, które pozwoli na wygenerowanie pary kluczy i stworzenie żądania certyfikacji (CSR). Przeglądarki kompatybilne z Netscape (Firefox, Chrome, etc.) oferują do tego celu tag **keygen** [3] będący regularnym elementem formularza. Microsoft odmówił jednak implementacji tego rozwiązania w Internet Explorer, ponieważ w systemach Windows Vista i nowszych dostępne jest *Certificate Enrollment API* [4], którego można użyć przez ActiveX z poziomu skryptu po stronie klienta.

Wobec tego, trzeba stworzyć aplikację, która zależnie od przeglądarki klienta zaserwuje mu formularz wykorzystujący kompatybilną technologię. Ze względu na ograniczoną ilość miejsca, implementacja omówiona w artykule będzie dosyć uproszczona, aby nie zagłębiać się w rzeczy mocno wykraczające poza temat.

Po stronie klienta przykład ograniczy się do dwóch statycznych stron z formularzami – jednym dla IE, drugim dla przeglądarek kompatybilnych z Netscape. Oba formularze będą zawierały pole do wpisania nicku, który zostanie później użyty jako *Common Name* (CN) w wygenerowanym certyfikacie, oraz kontrolkę generatora kluczy.

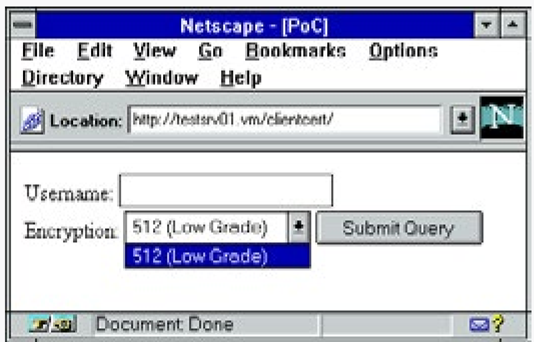
### Keygen

Element **<keygen>** reprezentuje generator pary kluczy i jest wyświetlany w przeglądarce analogicznie do tagu **<select>**, przy czym poszczególne opcje na liście są długościami kluczy wspieranymi przez dany silnik przeglądarki. Element ten posiada następujące unikalne atrybuty:

- **keytype** – algorytm klucza, który powinien zostać użyty (domyślnie **RSA**),
- **challenge** – ciąg znaków (domyślnie pusty), który zostanie podpisany i wysłany razem z żądaniem certyfikacji (prosta forma udowodnienia, że wysyłający żądanie jest równocześnie posiadaczem klucza).

Wysłanie formularza zawierającego kontrolkę generatora kluczy powoduje wygenerowanie żądania certyfikacji w formacie *Signed Public Key and Challenge* (SPKAC lub inaczej Netscape SPKI).

Oryginalnie w pełni funkcjonalny tag został zaimplementowany w 1996 roku w przeglądarce Netscape.




*Rysunek 3. Generator pary kluczy w Netscape 3.04 uruchomionym w systemie Windows 3.11*

Wynikowy formularz powinien wyglądać tak:

**Listing 1. Formularz netscape.html dla przeglądarek kompatybilnych z Netscape**

```
<form action="./certHandler/" method="POST">
    <label for="nick">Nick:</label>
    <input type="text" name="nick" value=""><br>

    <label for="csr">Encryption:</label>
    <keygen id="csr" name="csr">

    <input type="hidden" name="spki" value="spkac">
    <input type="submit" value="OK">
</form>
```

Przy próbie wysłania formularza żądanie certyfikacji zostanie automatycznie wygenerowane i wstawione jako parametr **csr**. W odpowiedzi na to zapytanie, serwer powinien wysłać wystawiony certyfikat wraz z nagłówkiem **Content-Type: application/x-x509-user-cert**

Przeglądarka automatycznie zainstaluje taki certyfikat w swoim magazynie.

### CertEnroll

Certificate Enrollment API to rozbudowany interfejs od Microsoft, który pozwala również na wystawianie żądań certyfikacji w formacie PKCS#10 oraz instalowanie wygenerowanego certyfikatu na komputerze klienta. Poprzez ActiveX z interfejsu można skorzystać za pomocą **X509EnrollmentWebClassFactory**[5]. Należy jednak pamiętać, że przeglądarka ze względów bezpieczeństwa nie pozwoli odwołać się do tego obiektu, jeśli strona z korzystającym z niego skryptem nie została pobrana przez HTTPS.

Pora na stworzenie formularza dla IE (plik *ie.html*). Pożądane byłoby, aby oba formularze miały podobny układ i wygląd. Wygląd tagu **<keygen>** można emulować, używając **<select>**.

**Listing 2. Formularz ie.html przypominający wyglądem formularz z pierwszego listingu**

```
<form id="csrForm" action="csrHandler"
    method="POST" onsubmit="return CreateReq();">
    <input type="hidden" name="spki" value="pkcs10">
    <input type="hidden" name="csr" id="csr">

    <label for="nick">Nick:</label>
    <input type="text" id="nick" name="nick" value=""><br>

    <label for="keysize">Encryption:</label>
    <select id="keysize">
        <option value="1024">1024 (Medium Grade)</option>
        <option value="2048">2048 (High Grade)</option>
    </select>
    <input type="submit" value="OK">
</form>
```

W tym wypadku żądanie certyfikacji nie zostanie wygenerowane automatycznie przy wysyłaniu formularza, cały proces trzeba przeprowadzić, tworząc odpowiedni skrypt JavaScript. Konieczne jest w tym miejscu zadeklarowanie obiektu fabryki, który posłuży do tworzenia potrzebnych obiektów CertEnroll. Do obsługi odpowiedzi serwera przyda się również jQuery.

**Listing 3. Potrzebne zależności oraz funkcja pomocnicza**

```
<object id="factory"
    classid="clsid:884e2049-217d-11da-b2a4-000e7bbb2b09"></object>
<script src="//ajax.googleapis.com/ajax/libs/jquery/1.10.2/
jquery.min.js">
</script>
<script type="text/javascript">
function CreateInst(name) {
    return factory.CreateObject("X509Enrollment." + name);
}
</script>
```

Ostatnim krokiem jest zdefiniowanie funkcji **CreateReq()**, która przy wysyłaniu formularza sprawdzi, jakiego wyboru co do długości klucza dokonał użytkownik, wygeneruje żądanie certyfikacji i wstawi je do pola **csr** w formularzu. Ponieważ instalacja certyfikatu również wymaga wywołania odwołania do obiektu **Enroll**, najwygodniejszym rozwiązaniem będzie wysłanie zawartości formularza AJAXem, a potem wywołanie **Enroll.InstallResponse** na otrzymanej odpowiedzi, co spowoduje instalację certyfikatu.

**Listing 4. Funkcja obsługująca wysłanie formularza**

```
function CreateReq() {
    try {
        var PrivKey = CreateInst("CX509PrivateKey");
        var Req = CreateInst("CX509CertificateRequestPkcs10");
        var Enroll = CreateInst("CX509Enrollment");

        // niezbędne stałe
        var ALLOW_UNTRUSTED_ROOT = 4;
        var AT_KEYEXCHANGE = 1;
        var CONTEXT_USER = 1;
        var XCN_CRYPT_STRING_BASE64 = 1;
        var XCN_CRYPT_STRING_BINARY = 2;

        // rozmiar klucza zgodnie z wyborem użytkownika
        PrivKey.Length = $("#keysize").val();
        PrivKey.KeySpec = AT_KEYEXCHANGE;
        Req.InitializeFromPrivateKey(CONTEXT_USER, PrivKey, "");

        // wystawiamy żądanie certyfikacji PKCS#10
        Enroll.InitializeFromRequest(Req);
        var pkcs10 = Enroll.CreateRequest(XCN_CRYPT_STRING_BASE64);

        $("#csr").val(pkcs10);

        $.post({
            url: "certHandler",
            data: $("#csrForm").serialize(),
            async: false,
            success: function(data) {
                Enroll.InstallResponse(ALLOW_UNTRUSTED_ROOT, data,
                    XCN_CRYPT_STRING_BINARY, "");
                alert("Zainstalowano certyfikat!");
            }
        });
    } catch(ex) {
        alert("Błąd: " + ex.description);
    }

    return false;
}
```

Należy zwrócić szczególną uwagę na fakt, że żądania PKCS#10 mogą nieść ze sobą DN podmiotu tworzącego żądanie. Nie ma jednak obowiązku dołączania tej informacji do żądania certyfikacji, a niekiedy wystawiający certyfikat może całkowicie ją zignorować i w odpowiedzi wystawić certyfikat o zupełnie innych parametrach.

### XEnroll

W systemie Windows XP wprowadzono interfejs *XEnroll*, który później został zdeprecjonowany i usunięty w następnej wersji. Z powodu złamania kompatybilności wstecznej nie jest możliwe użycie *XEnroll* w systemie innym niż XP, ale również nie jest możliwe użycie *CertEnroll* w systemie starszym niż Vista.

## Server-side: Servlet wystawiający certyfikaty

Zadaniem do zrealizowania po stronie serwera jest odbieranie żądań certyfikacji i wystawianie certyfikatów użytkownika. Te czynności będzie realizował **CertReqHandlerServlet**, zmapowany pod URL **/csrHandler** (adres docelowy dwóch wcześniej stworzonych formularzy). Ponadto, do manipulowania obiektami związanymi z certyfikatami wykorzystana zostanie biblioteka *BouncyCastle Crypto API* [6] (należy dodać do zależności pakiety **bcpkix-jdk15on** oraz **bcprov-jdk15on**).

Do wystawienia jakiegokolwiek certyfikatu potrzebny jest wcześniej stworzony certyfikat CA. Servlet zaimportuje ten certyfikat wraz z jego kluczem podczas inicjalizacji.

**Listing 5. Inicjalizacja servletu – ładowanie certyfikatu i klucza prywatnego**

```
private X509Certificate caCert;
private RSAPrivateKey caKey;

private Object readPEMObject(String path) throws IOException {
     BufferedReader br = new BufferedReader(new
FileReader(path));
     try {
        return new PEMParser(br).readObject();
     } finally {
        if (br != null) br.close();
     }
}

private void loadCACert(String path) {
    try {
       caCert = ClientCertUtil.unpackHolder(
          (X509CertificateHolder)readPEMObject(path));
    } catch (IOException e) {
       throw new RuntimeException("cannot load CA cert!", e);
    }
}

private void loadCAKey(String path) {
    try {
       PEMKeyPair pkPair = (PEMKeyPair)readPEMObject(path);
       caKey = (RSAPrivateKey)new JcaPEMKeyConverter()
          .getPrivateKey(pkPair.getPrivateKeyInfo());
    } catch(IOException e) {
       throw new RuntimeException("cannot load CA key!", e);
    }
}

public void init() throws ServletException {
    Security.addProvider(new BouncyCastleProvider());
    loadCACert(getServletContext()
       .getInitParameter("issuerPublicKey"));
    loadCAKey(getServletContext()
       .getInitParameter("issuerPrivateKey"));
}
```

Lokalizację certyfikatu CA oraz jego klucza (oba pliki w formacie PEM) precyzują parametry kontekstu **issuerPublicKey** oraz **issuerPrivateKey**. Powinny one wskazywać na pliki wygenerowane wcześniej za pomocą **openssl**.

Kolejnym elementem wymaganym do wystawienia certyfikatu X509 jest żądanie certyfikacji. W tym miejscu występują niekompatybilności – Internet Explorer przesyła na serwer żądania certyfikacji w formacie PKCS#10, podczas kiedy przeglądarki kompatybilne z Netscape dostarczają żądania jako SPKAC. Oba stworzone uprzednio formularze zawierają ukryty parametr **spki**, który określa rodzaj żądania certyfikacji.

**Listing 6. Metoda servletu wypakowująca z żądania certyfikacji informacje o kluczu publicznym**

```
private SubjectPublicKeyInfo resolveSPKI(String type, String csr)
       throws IOException {
    byte[] data = Base64.decode(csr);
    if (type.equals("pkcs10")) {
       return new PKCS10CertificationRequest(data)
          .getSubjectPublicKeyInfo();
    } else if (type.equals("spkac")) {
       return new SignedPublicKeyAndChallenge(data)
          .getPublicKeyAndChallenge()
          .getSubjectPublicKeyInfo();
    }

    throw new UnsupportedOperationException("unknown SPKI type");
}
```

Dostęp do większości strategicznych dla procesu wystawiania certyfikatu obiektów został już zapewniony. W przykładowej implementacji do wystawiania certyfikatów zostanie zdefiniowana klasa **ClientCertBuilder**.

**Listing 7. Konstruktor ustawiający domyślny algorytm podpisu**

```
private SubjectPublicKeyInfo spki;
private X509Certificate caCert;
private RSAPrivateKey caKey;
private String ownerCN;
private AlgorithmIdentifier sigAlgId;
private AlgorithmIdentifier digAlgId;

public ClientCertBuilder(SubjectPublicKeyInfo spki) {
    this.spki = spki;

    this.sigAlgId = new DefaultSignatureAlgorithmIdentifierFinder()
       .find("SHA1withRSA");
    this.digAlgId = new DefaultDigestAlgorithmIdentifierFinder()
       .find(sigAlgId);
}
```

Wszystkie pozostałe pola obiektu (**caCert**, **caKey** i **ownerCN**) powinny posiadać typowy, prosty setter. Do tworzonego certyfikatu należy zastosować kilka rozszerzeń X509.

**Listing 8. Metoda dodająca do certyfikatu wymagane rozszerzenia X509**

```
private X509v3CertificateBuilder addExtensions(
    X509v3CertificateBuilder builder)
       throws CertIOException, NoSuchAlgorithmException,
       CertificateEncodingException {
    // certyfikat końcowy (nie-CA)
    builder.addExtension(X509Extension.basicConstraints,
       false, new BasicConstraints(false));

    // identyfikator klucza podmiotu
    SubjectKeyIdentifier subjectIdentifier = new
    BcX509ExtensionUtils()
       .createSubjectKeyIdentifier(spki);
    builder.addExtension(X509Extension.subjectKeyIdentifier,
       false, subjectIdentifier);

    // identyfikator klucza CA, konieczny do prawidłowego
    // odnalezienia pasującego certyfikatu głównego przez Tomcata
    AuthorityKeyIdentifier identifier = new JcaX509ExtensionUtils()
       .createAuthorityKeyIdentifier(caCert);
    builder.addExtension(
       X509Extension.authorityKeyIdentifier, false,
       identifier);

    // przeznaczenie certyfikatu – uwierzytelnianie klienta
    builder.addExtension(X509Extension.extendedKeyUsage,
       false, new ExtendedKeyUsage(
       KeyPurposeId.id_kp_clientAuth));

    return builder;
}
```

Finalnie należy stworzyć metodę budującą certyfikat. W tym uproszczonym przykładzie metoda **createOwnerName** ustawia tylko dwa atrybuty – nazwę organizacji (skopiowaną z certyfikatu CA) oraz nazwę zwyczajową.

**Listing 9. Metoda build wraz z jej metodami pomocniczymi**

```
private X500Name createOwnerName(X500Name issuer) {
    return new X500NameBuilder(BCStyle.INSTANCE).addRDN(
       BCStyle.O, issuer.getRDNs(BCStyle.O)[0]
          .getFirst().getValue()
       ).addRDN(BCStyle.CN, ownerCN)
       .build();
}

private X500Name getCACertIssuer() {
    try {
       return new JcaX509CertificateHolder(caCert)
          .getSubject();
    } catch (CertificateEncodingException e) {
       throw new RuntimeException("failed to get issuer", e);
    }
}

public X509Certificate build(BigInteger serial, int days) {
    X500Name issuer = getCACertIssuer();
    X500Name owner = createOwnerName(issuer);

    Calendar cal = Calendar.getInstance();
    cal.add(Calendar.DATE, days);

    RSAKeyParameters params = new RSAKeyParameters(true,
       caKey.getModulus(), caKey.getPrivateExponent());

    ContentSigner sign;
    try {
       sign = new BcRSAContentSignerBuilder(sigAlgId,
          digAlgId).build(params);
    } catch (OperatorCreationException e) {
```

```
            throw new RuntimeException(
                "failed to build ContentSigner", e);
        }

        try {
            return ClientCertUtil.unpackHolder(addExtensions(
                new X509v3CertificateBuilder(issuer, serial,
                    new Date(), cal.getTime(), owner, spki)).build(sign));
        } catch (CertIOException
                | NoSuchAlgorithmException
                | CertificateEncodingException e) {
                throw new RuntimeException(
                    "failed to add " + "extensions", e);
        }
    }
```

Builder jest już gotowy, ostatnim krokiem będzie stworzenie metody wypluwającej wypisany certyfikat w odpowiedzi na żądanie klienta oraz zdefiniowanie metody **doPost** kontrolującej cały proces wystawiania certyfikatu.

**Listing 10. Metoda servletu odsyłająca wystawiony certyfikat klientowi**

```
    private void outputCert(HttpServletResponse res,
                            X509Certificate cert)
            throws IOException, CertificateEncodingException {
        res.setStatus(HttpServletResponse.SC_OK);
        res.setHeader("Connection", "close");
        res.setContentType("application/x-x509-user-cert");
        res.getOutputStream().write(cert.getEncoded());
    }
```

**Listing 11. Metoda servletu obsługująca wysyłane przez klientów formularze.**

```
    private void doPost(HttpServletRequest req,
                        HttpServletResponse res)
            throws IOException, ServletException {
        String spkiType = req.getParameter("spki");
        SubjectPublicKeyInfo spki = resolveSPKI(
            spkiType, req.getParameter("csr"));

        X509Certificate cert = new ClientCertBuilder(spki)
            .setCACert(caCert)
            .setCAKey(caKey)
            .setOwnerCN(req.getParameter("nick"))
            .build(new BigInteger("3"), 365);

        try {
            outputCert(res, cert);
        } catch (CertificateEncodingException e) {
            throw new ServletException("failed to encode certificate", e);
        }
    }
```

Aplikacja jest ukończona. Możliwe jest wystawianie certyfikatów zarówno użytkownikom z przeglądarką Internet Explorer, jak i użytkownikom pracującym w przeglądarkach kompatybilnych z Netscape. Należy jednak pamiętać, że przedstawiony przykład jest w niektórych miejscach ujęty w dużym uproszczeniu.

### Rozróżnianie użytkowników

Jeżeli użytkownik uwierzytelni swoje połączenie za pomocą certyfikatu użytkownika, zostanie ustawiony atrybut zapytania **javax.servlet.request.X509Certificate** zawierający tablicę obiektów **X509Certificate**. Pierwszym elementem jest zawsze certyfikat użytkownika, pozostałe elementy tej tablicy stanowią łańcuch wystawców. Użytkownika można rozpoznać na przykład poprzez *commonName* wpisany w jego certyfikacie.

## PODSUMOWANIE

Technologia certyfikatów użytkownika SSL zdecydowanie nie należy do najprostszych w implementacji, głównie przez jej stopień skomplikowania. Znajduje ona jednak zastosowanie w bardziej skomplikowanych systemach i jest bezpiecznym kryptograficznie rozwiązaniem umożliwiającym sprawdzenie wiarygodności obu stron podczas komunikacji.

### W sieci


- [1] http://tools.ietf.org/html/rfc5280
- [2] http://docs.oracle.com/javaee/7/tutorial/doc/servlets.htm
- [3] https://developer.mozilla.org/en-US/docs/Web/HTML/Element/keygen
- [4] http://msdn.microsoft.com/en-us/library/windows/desktop/aa374863(v=vs.85).aspx
- [5] http://msdn.microsoft.com/en-us/library/aa377863(v=vs.85).aspx
- [6] http://www.bouncycastle.org/docs/



**Michał Leszczyński**

redakcja@programistamag.pl

Freelancer z wieloletnim doświadczeniem w dziedzinie aplikacji webowych. Administrator systemów serwerowych na platformach UNIX-owych.

reklama