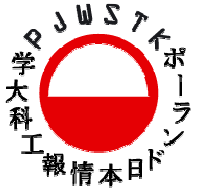


Polsko-Japońska Wyższa Szkoła Technik Komputerowych

WS-Security Framework

Edgar Głowacki
edgar@glowacki.eu

Współczesny szkielet (*framework*) warstwy pośredniej

- Oczekiwania względem szkieletu *middleware*
  - ✓ prawdziwa – a nie deklarowana – interoperacyjność
  - ✓ przejęcie odpowiedzialności za powtarzające kwestie wynikające z wymagań pozafunkcjonalnych: (1) przetwarzanie asynchroniczne, (2) bezpieczeństwo, (3) transakcyjność, …
- OMG CORBA – pierwsza próba stworzenia szkieletu *middleware* z prawdziwego zdarzenia – właściwie dogorywająca
- WebServices
  - ✓ znacznie więcej niż RPC – choć wiele osób mających małą wiedzę o WS widzi w tej technologii mało udany klon RPC
  - ✓ w rzeczywistości od roku 2001 WS były projektowane jako prawdziwy szkielet *middleware*

Model komunikacji SOAP

- Węzły SOAP
  - ✓ *original sender*
  - ✓ *intermediary*
  - ✓ *ultimate receiver*
- Przetwarzanie komunikatów przez węzły
  - ✓ bloki nagłówka – dodawane, usuwane, modyfikowane przez wszystkich uczestników komunikacji
  - ✓ ciało – komunikacja między końcami
- Atrybuty bloku nagłówka zdefiniowane przez SOAP
  - ✓ *role (actor SOAP 1.1)*
  - ✓ *mustUnderstand*
  - ✓ *relay*

3

Global XML Web Services Architecture

- Specyfikacje stanowiące „cegiełki” (*building blocks*) tworzące infrastrukturę szkieletu – zgodnie z paradygmatem *protocol composability*
- Specyfikacje WS-* (częściowo grupowane w szkielety)
  - ✓ szkielet bezpieczeństwa (WS-Security, WS-Trust, WS-Federation, WS-SecureConversation, WS-SecurityPolicy)
  - ✓ szkielet transakcyjności (WS-Coordination, WS-AtomicTransaction, WS-BusinessActivity) – posiada też konkurencyjny zbiór specyfikacji
  - ✓ szkielet niezawodnego transportu (WS-ReliableMessaging, …)
  - ✓ …
  - ✓ łącznie ok. 70-80 różnych specyfikacji na różnym etapie rozwoju

4

WS-Security – fundament szkieletu (1)

- Tunelowanie na poziomie transportu nieadekwatne dla modelu komunikacji SOAP
  - ✓ *point-to-point vs. end-to-end*
- Zabezpieczenie na poziomie komunikatu
  - ✓ nagłówek bezpieczeństwa **`<wsse: Security>`**
  - ✓ zawsze co najwyżej jeden nagłówek dla danego węzła
  - ✓ blok bez wskazanej roli może być przetwarzany przez wszystkie węzły, ale nie może być usunięty aż do osiągnięcia końca ścieżki
- poufność lub/i integralność
  - ✓ XML-Encryption i XML-Signature
  - ✓ zabezpieczanie dowolnej liczby wybranych elementów koperty SOAP

5

WS-Security – fundament szkieletu (2)

- Żetony bezpieczeństwa – jako bloki *security header* (określone w osobnych profilach)
  - ✓ nazwa użytkownika/hasło, certyfikat X.509, żeton Kerberos, asercja SAML
  - ✓ własne (*proprietary*) żetony bezpieczeństwa – oczywiście z podaną przestrzenią nazw
- Znaczniki czasu
  - ✓ mechanizm zabezpieczający przed *replay attack*
  - ✓ umożliwiają przekazanie informacji o czasie utworzenia i ważności elementów chronionych przez nagłówek bezpieczeństwa
  - ✓ nie jest to znacznik czasu w rozumieniu TSP (RFC 3161) – czas nie pochodzi z autorytatywnego źródła, ale jest lokalny dla strony wstawiającej blok nagłówka

6

WS-Security – fundament szkieletu (4)

```
<S12:Envelope ...>
  <S12:Header>
    ...
    <wsse:Security S12:mustUnderstand="false">
    </wsse:Security>
    ...
    <wsse:Security S12:role="ultimateReceiver"
      S12:mustUnderstand="true">
    </wsse:Security>
  </S12:Header>
</S12:Envelope>
```

źródło: [2]

WS-Security – fundament szkieletu (5)

```
<?xml version="1.0" encoding="utf-8"?>
<S:Envelope xmlns:S="..." xmlns:wsse="..." xmlns:ds.="..."
  <S:Header>
    <wsse:Security>
      <wsse:BinarySecurityToken
        ValueType="wsse:X509v3"
        EncodingType="wsse:Base64Binary"
        wsu:Id="X509Token">
        MIIEZzCCA9CgAwIBAgIQEmtJZc0rqrKh5i...
     </wsse:BinarySecurityToken>
     <ds:Signature>
       ..... <!-- i.a. metadata: canonicalization
       (normalization) method, signature schema, digest
       algorithm -->
       <ds:SignatureValue>BL8jdfToEb1l/vXcMZNNjPOV...
       </ds:SignatureValue> ....
     </ds:Signature>
    </wsse:Security>
  </S:Header>
  <S:Body wsu:Id="MsgBody">.....</S:Body>
</S:Envelope>
```

źródło: [1]

WS-Security – fundament szkieletu (6)

```
<!-- signature content compliant with XMLDSig extracted from
     sample security header -->
<ds:Signature>
  <ds:SignedInfo>
    <ds:CanonicalizationMethod
      Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
    <ds:SignatureMethod
      Algorithm="http://www.w3.org/2000/09/xmldsig#hmac-sha1"/>
    <ds:Reference URI="#MsgBody">
      <ds:DigestMethod
        Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
      <ds:DigestValue>LyLsF0Pi4wPU...</ds:DigestValue>
    </ds:Reference>
  </ds:SignedInfo>
  <ds:SignatureValue>DJbchm5gK...</ds:SignatureValue>
  <ds:KeyInfo>
    <wsse:SecurityTokenReference>
      <wsse:Reference URI="#MyID"/>
    </wsse:SecurityTokenReference>
  </ds:KeyInfo>
</ds:Signature>
```

źródło: [3]

WS-Policy – WSDL to za mało (1)

- Autonomiczność – jeden z głównych postulatów SOA
  - ✓ konieczność przekazania informacji innych wymaganiach – np. dotyczących sposobu kodowania komunikatu (np. MTOM), czy tego komu ufamy
- Szkielet WS-Policy
  - ✓ kompozycja polityki – asercje (atomowe warunki) i alternatywy konstruowane w oparciu o operatory: *ExactlyOne*, *All*, *OneOrMore*
  - ✓ definicja polityki może być umieszczona w WSDL, bądź też WSDL może zawierać referencję do niej
  - ✓ różne zakresy (podmioty) obowiązywania polityki (punkt końcowy, komunikat, konwersacja, …) – WS-PolicyAttachment
  - ✓ generyczny szkielet uzupełniany przez odpowiednie specyfikacje (WS-PolicyAssertions, WS-SecurityPolicy, …)

WS-Policy – WSDL to za mało (2)

```
<!-- sample policy definition compliant with WS-Policy -->
<wsp:Policy
  xmlns:sp="http://docs.oasis-open.org/ws-sx/ws-
  securitypolicy/200702"  <!-- WS-SecurityPolicy defines
                              security assertions -->
  xmlns:wsp="http://www.w3.org/ns/ws-policy">
  <wsp:ExactlyOne>          <!-- exactly one operator -->
    <wsp:All>               <!-- first alternative -->
      <sp:SignedParts>
        <sp:Body/>          <!-- body must be either signed -->
      </sp:SignedParts>
    </wsp:All>
    <wsp:All>
      <sp:EncryptedParts>
        <sp:Body/>          <!-- or encrypted -->
      </sp:EncryptedParts>
    </wsp:All>              <!-- second alternative -->
  </wsp:ExactlyOne>
</wsp:Policy>
```

źródło: [8]

WS-Policy – WSDL to za mało (3)

```
<!-- associating policy with SOAP endpoint -->
...
<wsp:PolicyAttachment>
  <wsp:AppliesTo>
    <wsp:EndpointReference>
      <wsp:ServiceName Name="InventoryService"/>
      <wsp:PortType Name="InventoryPortType"/>
      <wsp:Address URI="http://www.xyz.com/acct"/>
    </wsp:EndpointReference>
  </wsp:AppliesTo>
  <wsp:PolicyReference
    Ref="http://www.xyz.com/acct-policy.xml"/>
</wsp:PolicyAttachment>
```

źródło: [1]

WS-SecurityPolicy – asercje bezpieczeństwa (1)

- Asercje określające sposób ochrony (wskazanych elementów) komunikatów
  - ✓ *integrity assertion*
  - ✓ *confidentiality assertion*
- Asercje określające wymagania względem żetonów
  - ✓ *token inclusion* – dołączenie żetonu do komunikatu
  - ✓ *token issuer* – zaufana trzecia strona
  - ✓ wymaganie dotyczące rodzajów żetonów i sposobu ich uzyskania: *UsernameToken, X509Token, SecurityConversationToken*

13

WS-SecurityPolicy – asercje bezpieczeństwa (2)

```
<wsp:Policy ...>
  <wsp:ExactlyOne>
    <!-- X.509 certificate is preferred over Kerberos token -->
    <wsse:SecurityToken TokenType="wsse:X509v3"
          wsp:Usage="wsp:Required" wsp:Preference="50"/>
    <wsse:SecurityToken TokenType="wsse:Kerberosv5TGT"
          wsp:Usage="wsp:Required" wsp:Preference="10"/>
  </wsp:ExactlyOne>
</wsp:Policy>
```

źródło: [1]

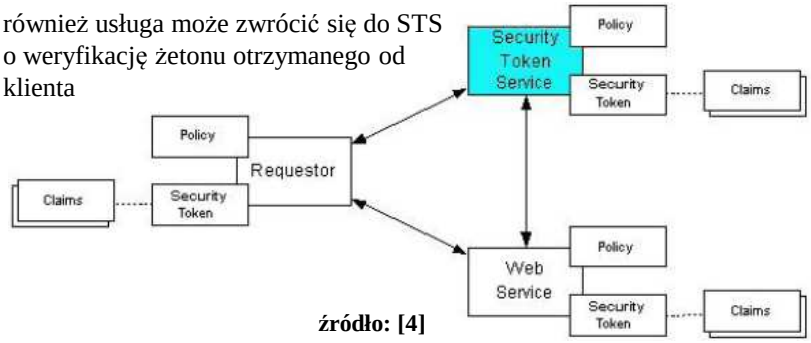
```
...
<wsp:Policy ...>
  <sp:SecureConversationToken>
    <sp:Issuer>
      <wsa:Address>http://example.org/sts</wsa:Address>
    </sp:Issuer>
    ...
  </sp:SecureConversationToken>
</wsp:Policy>
...
```

źródło: [3]

14

WS-Trust – budowa relacji zaufania (1)

- Model zaufania WS – wzorowany na Kerberos
  - ✓ w polityce usługa może zażądać by żądający (*requestor*) przedstawił w komunikacie dowody swoich uprawnień (roszczeń) (*proof of claims*), które powinny być zawarte w żetonie bezpieczeństwa
  - ✓ jeśli klient (*requestor*) takowych nie posiada może uzyskać je od STS
  - ✓ STS może przeprowadzić uwierzytelnienie i autoryzację
  - ✓ również usługa może zwrócić się do STS o weryfikację żetonu otrzymanego od klienta



źródło: [4]

15

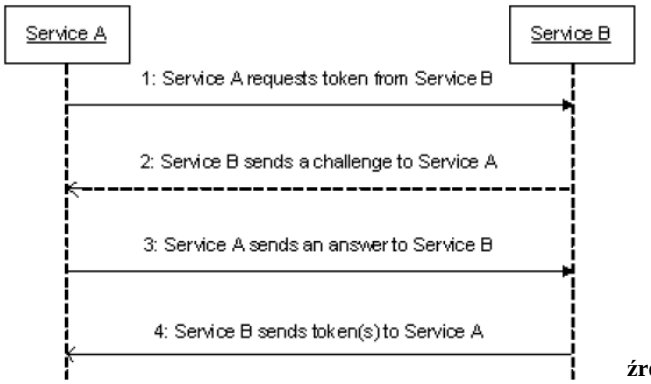
WS-Trust – budowa relacji zaufania (2)

- Przykłady wariantów architektury zaufania zgodne z WS-Trust
  - ✓ STS sam może być był usługą z określoną polityką wymagającą przedstawienia żetonów bezpieczeństwa
  - ✓ w architekturze zaufania wykorzystującej pośrednika (*brokered trust*) żeton może być uzyskany od STS a następnie dołączony do nagłówka przez węzeł pośredniczący
- Funkcjonalność STS (przykładowy WSDL [5])
  - ✓ wystawienie żetonu lub kolekcji żetonów
  - ✓ walidacja
  - ✓ anulowanie
  - ✓ odnowienie

16

WS-Trust – budowa relacji zaufania (3)

- Możliwa wymiana *challenge/response* z usługą STS poprzedzająca wystawienie żetonu
  - ✓ *signature challenge*
  - ✓ *user interaction challenge*



źródło: [4]

17

WS-Trust – *sample signature challenge* (1)

```
<!-- initial request signed with private key corresponding to
     attached certificate -->
<S11:Envelope ...>  <S11:Header> ... <wsse:Security>
      <wsse:BinarySecurityToken wsu:Id="reqToken"
        ValueType="...X509v3">MIIEZzCCA9C...
      </wsse:BinarySecurityToken> <!-- X.509 token -->
      <ds:Signature ...> ... <!-- signature issued with -->
        <ds:KeyInfo>          <!-- corresponding private key -->
          <wsse:SecurityTokenReference>
            <wsse:Reference URI="#reqToken"/>
          </wsse:SecurityTokenReference>
        </ds:KeyInfo>
      </ds:Signature>
   </wsse:Security> ... </S11:Header>
  <S11:Body>
    <wst:RequestSecurityToken>
      <wst:TokenType>http://example.org/token</wst:TokenType>
      <wst:RequestType>http://docs.oasis-open.org/ws-sx/ws-
trust/200512/Issue</wst:RequestType>
    </wst:RequestSecurityToken>
  </S11:Body> </S11:Envelope>
```

źródło: [4]

18

WS-Trust – *sample signature challenge* (2)

```
<!-- to prevent replay attack (e.g. timestamp has not been
     included in the request) STS issues a challenge to be
     signed by requestor -->
<S11:Envelope ...> <S11:Header> ... <wsse:Security>
      <wsse:BinarySecurityToken wsu:Id="issuerToken"
        ValueType="...X509v3„>DFJHue...
      </wsse:BinarySecurityToken>
      <ds:Signature ...> ... </ds:Signature>
    </wsse:Security> ... </S11:Header>
  <S11:Body>
    <wst:RequestSecurityTokenResponse>
      <wst:SignChallenge> <!-- sign the below challenge with
                               the key endorsed by the
                               trusted third party -->
        <wst:Challenge>Huehf...</wst:Challenge>
      </wst:SignChallenge>
    </wst:RequestSecurityTokenResponse>
  </S11:Body> ... </S11:Envelope>
```

źródło: [4]

19

WS-Trust – *sample signature challenge* (3)

```
<!-- in reply the requestor signs the message as specified in
     WS-Security -->
<S11:Envelope ...> <S11:Header> ... <wsse:Security>
      <wsse:BinarySecurityToken wsu:Id="reqToken"
        ValueType="...X509v3">MIIEZ...
      </wsse:BinarySecurityToken>
      <ds:Signature xmlns:ds="..."> ... </ds:Signature>
    </wsse:Security> ... </S11:Header>
  <S11:Body>
    <wst:RequestSecurityTokenResponse>
       <wst:SignChallengeResponse>
         <wst:Challenge>Huehf...</wst:Challenge>
       </wst:SignChallengeResponse>
    </wst:RequestSecurityTokenResponse>
  </S11:Body>
</S11:Envelope>
```

źródło: [4]

20

WS-Trust – *sample signature challenge* (4)

```
<!-- STS returns a collection of requested tokens:
     (1) a custom token and (2) a shared key -->
<S11:Envelope ...> <S11:Header> ... <wsse:Security>
      ...
    </wsse:Security> ... </S11:Header>
  <S11:Body>
    <wst:RequestSecurityTokenResponseCollection>
      <wst:RequestSecurityTokenResponse>
        <wst:RequestedSecurityToken>
          <xyz:CustomToken ...> ... </xyz:CustomToken>
        </wst:RequestedSecurityToken>
        <wst:RequestedProofToken>
          <xenc:EncryptedKey Id="newProof„>
            ...
          </xenc:EncryptedKey>
        </wst:RequestedProofToken>
      </wst:RequestSecurityTokenResponse>
    </wst:RequestSecurityTokenResponseCollection>
  </S11:Body>
</S11:Envelope>
```

źródło: [4]

21

WS-Trust – *user interaction challenge* (1)

```
<!-- initial request containing username and password in
     plaintext -->
<S11:Envelope ...> <S11:Header> ... <wsse:Security>
      <wsse:UsernameToken>
        <wsse:Username>Zoe</wsse:Username>
        <wsse:Password Type="http://...#PasswordText">
          ILoveDogs
        </wsse:Password>
      </wsse:UsernameToken>
    </wsse:Security> ... </S11:Header>
  <S11:Body>
    <wst:RequestSecurityToken>
      <wst:TokenType>
        http://example.org/customToken
      </wst:TokenType>
      <wst:RequestType>...</wst:RequestType>
    </wst:RequestSecurityToken>
  </S11:Body>
</S11:Envelope>
```

źródło: [4]

22

WS-Trust – *user interaction challenge* (2)

```
<!-- STS sends a challenge for a PIN -->
<S11:Envelope ...>
  <S11:Header> ... </S11:Header>
  <S11:Body>
    <wst:RequestSecurityTokenResponse>
      <wst14:InteractiveChallenge xmlns:wst14="..." >
        <wst14:TextChallenge
          RefId=http://docs.oasis-open.org/ws-sx/ws-
trust/200802/challenge/PIN
          Label="Please enter your PIN"/>
        </wst14:TextChallenge>
      </wst14:InteractiveChallenge>
    </wst:RequestSecurityTokenResponse>
  </S11:Body>
</S11:Envelope>
```

źródło: [4]

WS-Trust – *user interaction challenge* (3)

```
<!-- requestor application displays dialog soliciting PIN and
     sends it back to the STS afterward -->
<S11:Envelope ...>
  <S11:Header> ... </S11:Header>
  <S11:Body>
    <wst:RequestSecurityTokenResponse>
      <wst14:InteractiveChallengeResponse xmlns:wst14="..." >
        <wst14:TextChallengeResponse
          RefId="http://docs.oasis-open.org/ws-sx/ws-
trust/200802/challenge/PIN">
          9988
        </wst14:TextChallengeResponse>
      </wst14:InteractiveChallengeResponse>
    </wst:RequestSecurityTokenResponse>
  </S11:Body>
</S11:Envelope>
```

źródło: [4]

WS-Trust – zastosowanie żetonu określa polityka

- Schemat uwierzytelnienia dostarczy nam żeton wymagany przez politykę usługi
- Żeton jest następnie dołączony do nagłówka bezpieczeństwa komunikatu przekazanego odbiorcy, którym może być:
  - ✓ *ultimate receiver*
  - ✓ *intermediary*

25

WS-Federation – federacja domen zaufania WS

- Integracja przekraczająca granice domen zaufania (*trust realms*) oparta o współdzielenie
  - ✓ dowodów tożsamości (*identities*)
  - ✓ dodatkowej informacji o żądającym (np. imię i nazwisko) na zasadach regulujących federację i z uwzględnieniem zasad prywatności. (Przykładowo podczas korzystania ze sfederowanej usługi ujawniana jest jedynie informacja, że żądającym jest pracownik instytucji partnerskiej – bez podawania danych osobistych)
  - ✓ tzw. meta-danych federacji

26

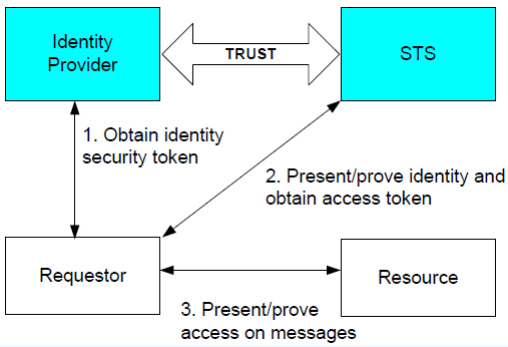
WS-Federation – meta-dane federacyjne usługi

- Meta-dane federacyjne usługi
  - ✓ opis sposobu wykorzystania usługi w ramach federacji – w tym przede wszystkim określenie roli: (1) usługa docelowa – zasób (*resource*), (2) STS, (3) *identity provider*, (4) *attribute service*, (5) *pseudonym service*
  - ✓ uzupełnienie WSDL i polityki – podobnie jak polityka mogą być dołączane do usługi na zasadach określonych WS-PolicyAttachment
  - ✓ podobnie jak polityka mogą mieć różny zasięg (*scope*) określany przez WS-PolicyAttachment: usługę, punkt końcowy, czy komunikat
  - ✓ podobnie jak polityka meta-dane federacji mogą być dostępne za pomocą WS-MetadataExchange (WS-MEX)
  - ✓ rekurencyjny dostęp do meta-danych usług powiązanych poprzez referencje – konieczne do określenia pełnych wymagań usługi

27

WS-Federation – *Identity Provider (IP)*

- *Identity Provider*
  - ✓ usługa dostarczająca żetony bezpieczeństwa potwierdzające tożsamość klienta względem usługi
  - ✓ odpowiednik *Key Distribution Center (Authentication Server + Ticket Granting Service*) znanego z protokołu Kerberos
  - ✓ usługa IP może być łączona z lokalnym STS (na diagramie przedstawiono zdalny STS)



źródło: [6]

28

WS-Federation – model bezpieczeństwa

- Model bezpieczeństwa
  - ✓ meta-dane federacyjne umożliwiają klientowi (*requestor*) uzyskanie informacji o rodzajach żetonów wymaganych przez usługę docelową
  - ✓ zgodnie z WS-Trust klient ubiega się o tzw. „początkowy” żeton identyfikujący (*initial security token*) u lokalnego IP/STS
  - ✓ żeton identyfikujący pozwala na uzyskanie żetonów u IP/STS obcych domen
  - ✓ żeton identyfikujący może również bezpośrednio umożliwić korzystanie z usług w innych domenach zaufania

29

WS-Federation – kontrola prywatności

- Zapewnienie prywatności w ramach federacji
  - ✓ kontrola dostępu do danych osobowych przez klienta
  - ✓ używanie względnie losowych identyfikatorów (pseudonimów – patrz. dalej) w celu utrudnienia niepożądanej korelacji tożsamości używanych w ramach różnych domen zaufania

30

WS-Federation – *Attribute Service (AS)*

- *Attribute Service*
  - ✓ dodatkowa informacja o kliencie (*requestor*) może być potrzebna w celu realizacji żądania, bądź personalizacji
  - ✓ możliwość dostępu do danych wrażliwych o kliencie tylko po odpowiedniej autoryzacji z jego strony

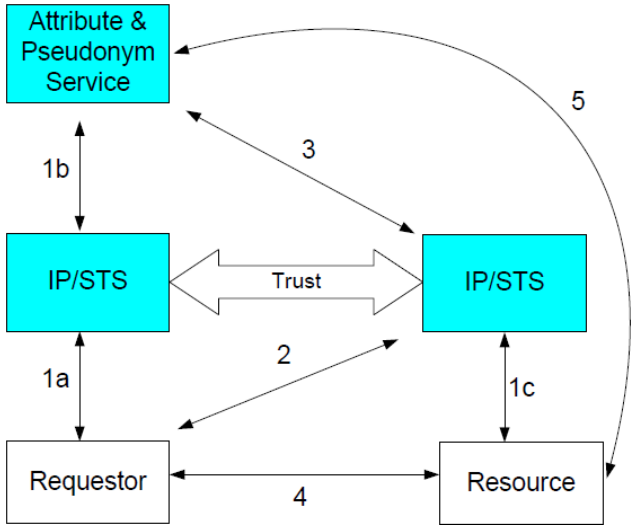
31

WS-Federation – *Pseudonym Service (PS)*

- *Pseudonym Service*
  - ✓ konieczność automatycznego mapowania tożsamości używanych w różnych domenach
  - ✓ usługa PS umożliwia danemu podmiotowi (*principal*) używanie różnych aliasów w czasie dostępu do poszczególnych domen, bądź zasobów
  - ✓ czas istnienia pseudonimów może być dowolny: (1) na stałe, (2) na czas interakcji (sesji) użytkownika, lub (3) pojedynczego dostępu do usługi
  - ✓ mapowanie identyfikatorów może być przeprowadzone przez (1) IP/STS w trakcie tworzenia żetonu na potrzeby dostępu do usługi, (2) przez samą usługę, bądź też (3) klienta – po otrzymaniu żetonu dla usługi

32

WS-Federation – federacja domen zaufania WS (7)



źródło: [6]

33

WS-Federation – federacja domen zaufania WS (8)

1a) Żądający uwierzytelnia się względem IP/STS własnej domeny

2) Żądający komunikuje się z IP/STS zasobu w celu uzyskania żetonu umożliwiającego dostęp do zasobu

3) IP/STS zasobu zarejestrował pseudonim żądającego (np. na czas dostępu do zasobów danej domeny). Pseudonim może być uzupełniony o odpowiednie atrybuty dostarczone przez żądającego – z odpowiednimi obostrzeniami związanymi z autoryzacją dostępu.

4) Żądający korzysta z usługi

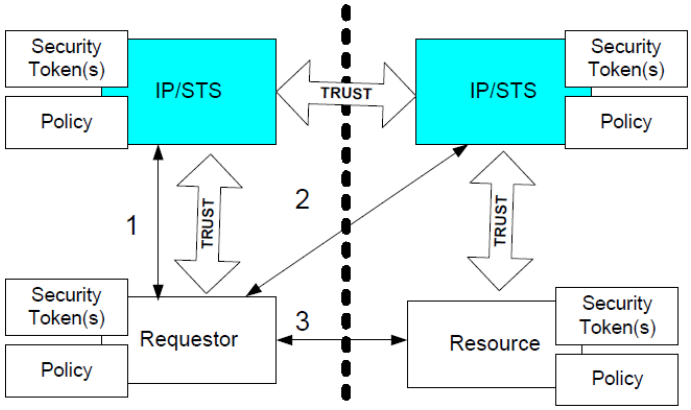
5) Usługa korzysta z atrybutów żądającego – po wcześniejszej autoryzacji u swojego IP/STS (1c)

1b) IP/STS żądającego rejestruje atrybuty klienta lub też IP/STS żądającego pobiera wcześniej zarejestrowany pseudonim żądającego – wówczas może nie być potrzeby wykonania kroku (2) i (3)

34

WS-Federation – topologie zaufania (1)

Scenariusz najprostszy



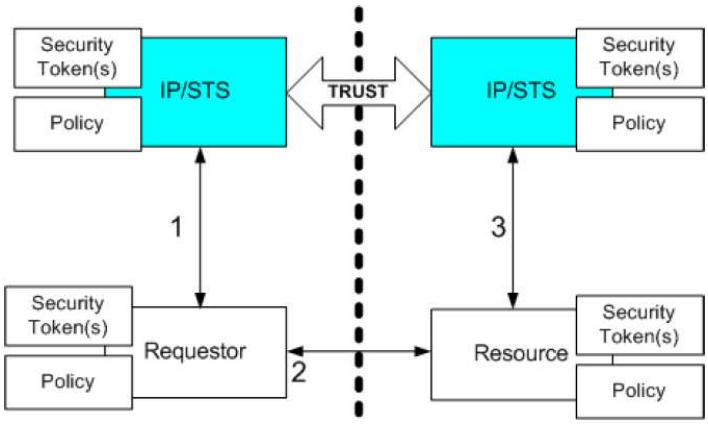
źródło: [6]

W kroku 2. obcy STS może wydać nowy żeton, bądź też odpowiednio certyfikować żeton uzyskany w kroku 1.

35

WS-Federation – topologie zaufania (2)

Usługa docelowa przedstawia obcy żeton własnemu STS w celu walidacji i autoryzacji.

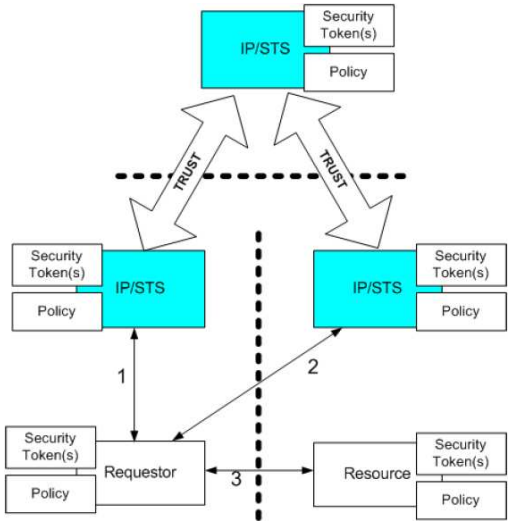


źródło: [6]

36

WS-Federation – topologie zaufania (3)

Relacja zaufania może być przechodnia



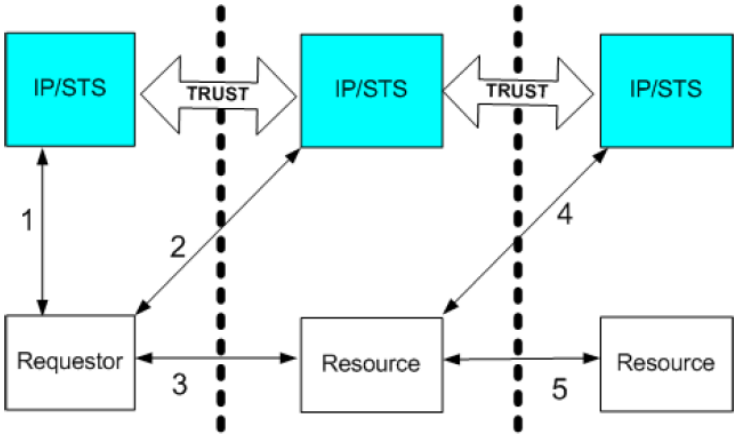
źródło: [6]

37

WS-Federation – rzeczywiste scenariusze (1)

Zasób wywołuje usługę spoza własnej domeny w imieniu żądającego.

Zasób może – ale nie musi – pełnić w tym przypadku rolę węzła pośredniczącego.

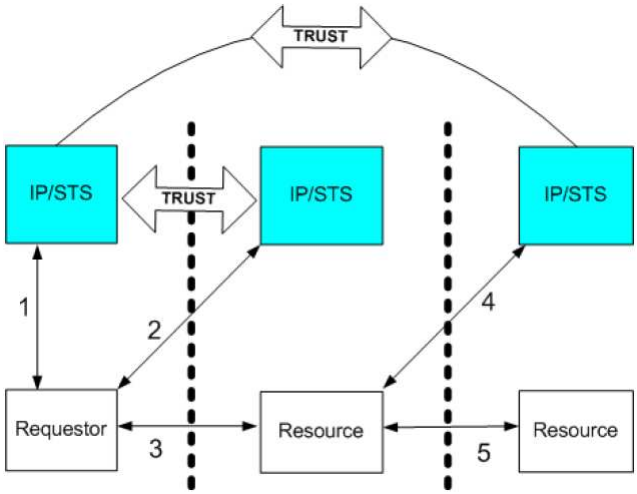


źródło: [6]

38

WS-Federation – rzeczywiste scenariusze (1)

Wariant poprzedniego scenariusza – klient delegował swoje uprawnienia na usługę pośredniczącą. Zasób bezpośrednio wywoływany może pełnić rolę *intermediary*.



źródło: [6]

39

WS-SecureConversation – zabezpieczanie sesji (1)

- Problem wydajności WS-Security
  - ✓ zabezpieczanie każdego komunikatu z osobna
  - ✓ potrzeba wprowadzenia sprawdzonego mechanizmu jakim jest kontekst bezpieczeństwa (znanego z SSL/TLS, IPSec, czy SSH) zawierający m.in. współdzielony klucz sesyjny
- WS-SecureConversation
  - ✓ rozszerzenie WS-Security wykorzystujące mechanizmy WS-Trust
  - ✓ *SecurityContextToken* – nowy rodzaj żetonu umieszczanego w nagłówku bezpieczeństwa zawierający unikalny identyfikator kontekstu wykorzystywanego przez strony konwersacji

40

WS-SecureConversation – zabezpieczanie sesji (2)

- Ustalanie i dystrybucja kontekstu bezpieczeństwa
  - ✓ wymaga interakcji przy użyciu odpowiedniego protokołu *challenge/response* przed rozpoczęciem sesji
  - ✓ API zbliżone do STS opisane w WS-Trust
  - ✓ komunikacja związana z wystawieniem i odnowieniem kontekstu oparta o schemat WS-Trust [7]
  - ✓ dystrybuowany tajny klucz sesyjny jest zawsze opakowywany przez element *wst:RequestedProofToken*
- Sposoby ustalania kontekstu bezpieczeństwa
  - ✓ STS – w oparciu o API WS-Trust [5]
  - ✓ jedna ze stron ustala i rozpowszechnia żeton – pozostali mogą go odrzucić
  - ✓ negocjacja – zależna od konkretnego rozwiązania (zaleca się schemat WS-Trust)

41

Podsumowanie (1)

- WS – względnie udana próba zapewnienia interoperacyjności
  - ✓ WS są powszechnie stosowane
  - ✓ niestety wykorzystanie WS jest najczęściej ograniczone do specyfikacji objętych przez profil interoperacyjności WS-I Basic Profile (WS-I BP) – i to też nie w ostatniej wersji (1.2) obejmującej WS-Addressing
  - ✓ WS-I BP znakomicie obrazuje stopień adopcji WS przez rynek – najnowsza wersja 1.2 poza SOAP, WSDL i UDDI zawiera też MTOM i WS-Addressing

42

Podsumowanie (2)

- WS-* powoli zyskują na popularności
  - ✓ specyfikacje dojrzewają – proces powstawania WS-* jest równie chaotyczny i burzliwy jak niegdyś prace nad OMG CORBA
  - ✓ użycie WS-* staje się coraz powszechniejsze ze względu na wsparcie ze strony implementacji: Microsoft WCF (wcześniej eksperyment WSE), rozszerzenia Axis2, ...
  - ✓ WS-* coraz częściej wykorzystywane jest również w produktach (np. w realizacji SSO przez Novell AC 3.1 w dostępie użytkowników niekorzystających z produktów Microsoft do witryn wykorzystujących Windows Authentication Provider)

Źródła informacji

[1] http://www.developer.com/services/article.php/2171031

[2] http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702/ws-securitypolicy-1.2-spec-os.pdf

[3] http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf

[4] http://docs.oasis-open.org/ws-sx/ws-trust/v1.4/ws-trust.pdf

[5] http://docs.oasis-open.org/ws-sx/ws-trust/v1.4/cd/ws-trust-1.4.wsdl

[6] http://docs.oasis-open.org/wsfed/federation/v1.2/os/ws-federation-1.2-spec-os.pdf

[7] http://docs.oasis-open.org/ws-sx/ws-trust/v1.4/ws-trust.xsd

[8] WS-Policy Framework 1.5

Dziękuję za uwagę

45