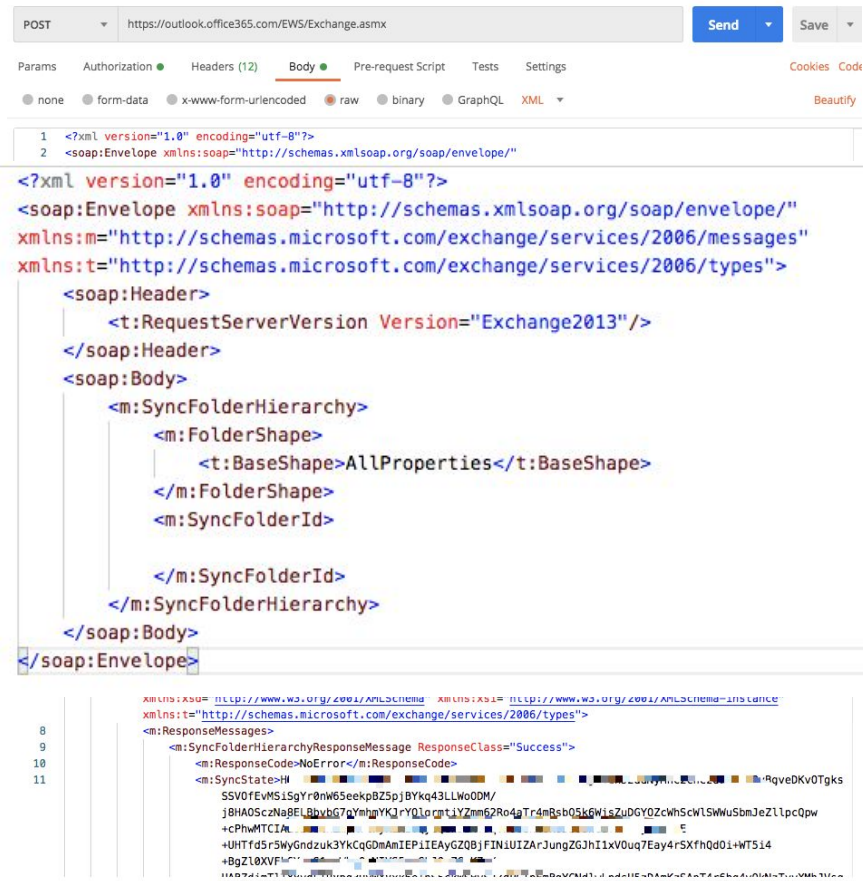


Verifying PolicyBased Security for Web Services

CS 563 / ECE 524 Spring 2021
Zhenyu Mao

SOAP



```
POST https://outlook.office365.com/EWS/Exchange.asmx Send Save

Params Authorization Headers (12) Body Pre-request Script Tests Settings Cookies Code

none form-data x-www-form-urlencoded raw binary GraphQL XML Beautify

1 <?xml version="1.0" encoding="utf-8"?>
2 <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
  <?xml version="1.0" encoding="utf-8"?>
  <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:m="http://schemas.microsoft.com/exchange/services/2006/messages"
    xmlns:t="http://schemas.microsoft.com/exchange/services/2006/types">
      <soap:Header>
        <t:RequestServerVersion Version="Exchange2013"/>
      </soap:Header>
      <soap:Body>
        <m:SyncFolderHierarchy>
          <m:FolderShape>
            <t:BaseShape>AllProperties</t:BaseShape>
          </m:FolderShape>
          <m:SyncFolderId>
            </m:SyncFolderId>
          </m:SyncFolderHierarchy>
        </soap:Body>
      </soap:Envelope>
    </m:SyncFolderHierarchyResponseMessage ResponseClass="Success">
      <m:ResponseCode>NoError</m:ResponseCode>
      <m:SyncState>H
        SSV0FEVMS15gYr0nw5Seekp0Z5pJBYkq43LLWo0DM/
        jBHA0Sc2NaBELBbybG7qYmhmYKJrYQLogmUjYZnm6ZRo4pTr4mRsb05k6W1sZuDGy0ZcWhScfLSWwU5bmJeZllpcQw
        +cPhwMTCIA
        +UHTfd5r5WyGndzuk3YkCqGdmAmIEPlIEAyGZQBjFINiUIZArJungZGJH1xV0uq7Eay4rSXfhQd0i+WT514
        +BgZL0XVF
      </m:SyncState>
    </m:SyncFolderHierarchyResponseMessage>
  </soap:Envelope>
```

SOAP Message

- XML Document
- Used to exchange structured information in web services

Is SOAP still in use?

- Outlook (EWS, EAS)
- Bank System
- Telecommunication
- Government API [NOAA, etc.]

Compare with RESTful (eg. JSON)?

WS-Security and XML Rewriting Attack

```
<Envelope>
  <Header>
    <From>http://www.client.com</ >
    <To>http://www.stockquote.com</ >
    <Attack>
      <To id="id4">http://www.airlineticket.com</ >
    </ >
    <Security mustUnderstand="1">
      <BinarySecurityToken Id="Id-2">abcdefg....</ >
      <Signature>
        <SignedInfo>
          <Reference URI="#Id-4">
            <DigestMethod Algorithm="#sha1" />
            <DigestValue>4AFDE67...</ ></ >
          <Reference URI="#Id-3">
            <DigestMethod Algorithm="#sha1" />
            <DigestValue>4AFDE67...</ ></ >
          </ >
          <SignatureValue>34EADB98...</ >
        </ >
        <KeyInfo>
          <SecurityTokenReference>
            <Reference URI="#Id-2" /></ >
          </ >
        </ >
      </ >
    </ >
  </ >
  <Body Id="Id-3">
    <AirlineTicketRequest>...</ >
  </ >
</ >
```

Main goal of WS-Security:

- sign, encrypt SOAP messages
- attach security tokens to ascertain the sender's identity

What is XML Rewriting Attack?

- Adding new elements to the SOAP header without compromising the contents of the message.

← Redirection Attack

- Change something out of protect

WS-Security and XML Rewriting Attack

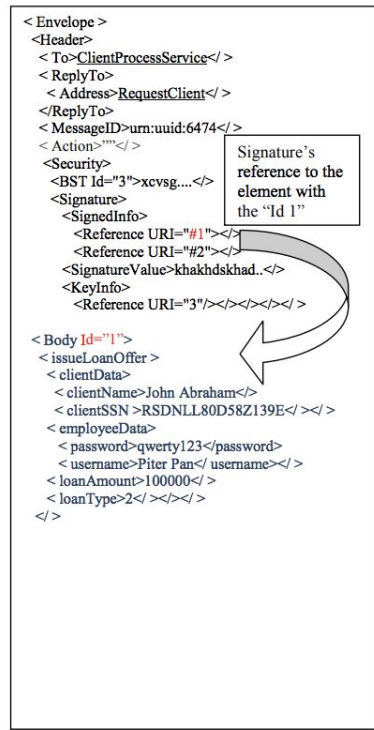


Figure 2. SOAP Message

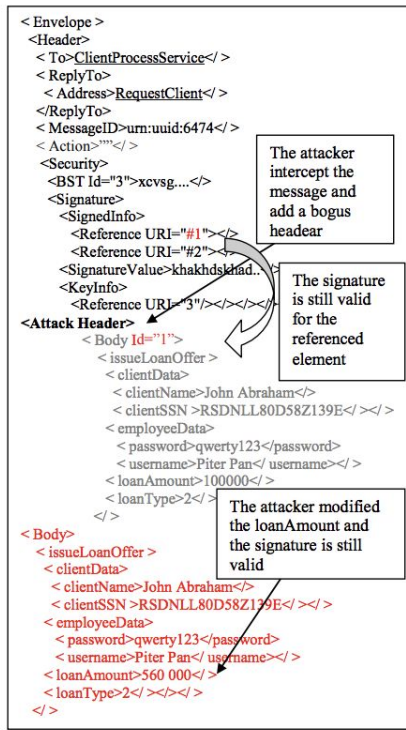


Figure 3. SOAP message after XML rewriting attack

Main goal of WS-Security:

- sign, encrypt SOAP messages
- attach security tokens to ascertain the sender's identity

What is XML Rewriting Attack?

- Adding new elements to the SOAP header without compromising the contents of the message.

← Reply Attack

- Change the body (loan money)

WS-Security and XML Rewriting Attack

Main goal of WS-Security:

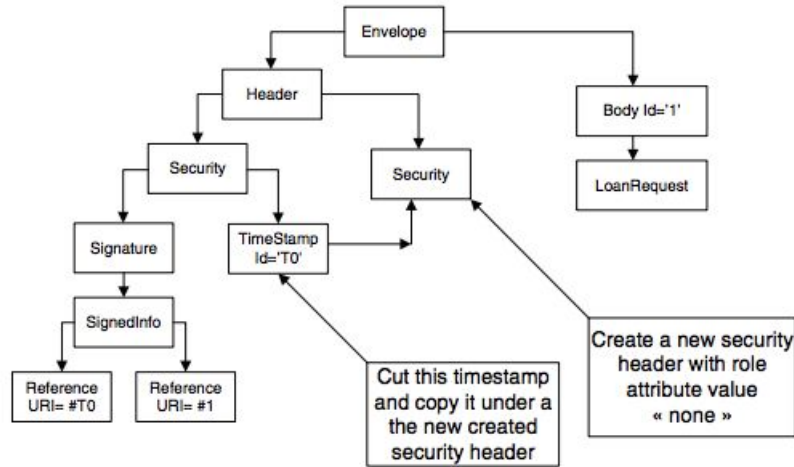
- sign, encrypt SOAP messages
- attach security tokens to ascertain the sender's identity

What is XML Rewriting Attack?

- Adding new elements to the SOAP header without compromising the contents of the message.

← Multiple Security Header attack

- Override the origin header



Defense: SOAPAccount

```
<Envelope>
<Header>
  <Security>
    <UsernameToken Id=3>
      <Username>Alice</>
      <Nonce>cGxr8w2AnBUzuhLzDYDoVw==</>
      <Created>2003-02-04T16:49:45Z</>
    </UsernameToken>
    <Signature>
      <SignedInfo>
        <Reference URI= #1>
          <DigestValue>Ego0...</>
        </Reference>
        <Reference URI= #2>
          <DigestValue>Qser99...</>
        </Reference>
        <Reference URI= #3>
          <DigestValue>OUytt0...</>
        </Reference>
        <SignatureValue>
          vSB9JU/Wr8ypAlaxCx2KdvjZcc=</>
        </SignatureValue>
      </SignedInfo>
      <KeyInfo>
        <SecurityTokenReference>
          <Reference URI=#3/>
        </SecurityTokenReference>
      </KeyInfo>
    </Signature>
  </Security>
  <SoapAccount id=2>
    <NoChildOfEnvelope>2</>
    <NoOfHeader>
      </SoapAccount>
    </NoOfHeader>
  </SoapAccount>
</Header>
<Body Id=1>
  <TransferFunds>
    <beneficiary>Bob</>
    <amount>1000</>
  </TransferFunds>
</Body>
</Envelope>
```

Message to bank's web service says: "Transfer 1000 euro to Bob, signed Alice"

Verifying signature using key derived from Alice's secret password

Fig. 9. A SOAP request before an attack (Excerpt)

```
<Envelope>
<Header>
  <Security>
    <UsernameToken Id=3>
      <Username>Alice</>
      <Nonce>cGxr8w2AnBUzuhLzDYDoVw==</>
      <Created>2003-02-04T16:49:45Z</>
    </UsernameToken>
    <Signature>
      <SignedInfo>
        <Reference URI= #1>
          <DigestValue>Ego0...</>
        </Reference>
        <Reference URI= #2>
          <DigestValue>Qser99...</>
        </Reference>
        <Reference URI= #3>
          <DigestValue>OUytt0...</>
        </Reference>
        <SignatureValue>
          vSB9JU/Wr8ypAlaxCx2KdvjZcc=</>
        </SignatureValue>
      </SignedInfo>
      <KeyInfo>
        <SecurityTokenReference>
          <Reference URI=#3/>
        </SecurityTokenReference>
      </KeyInfo>
    </Signature>
  </Security>
  <SoapAccount id=2>
    <NoChildOfEnvelope>2</>
    <NoOfHeader> 2 </>
  </SoapAccount>
  <BogusHeader>
    <Body Id=1>
      <TransferFunds>
        <beneficiary>Bob</>
        <amount>1000</>
      </TransferFunds>
    </Body>
  </BogusHeader>
  <Body>
    <TransferFunds>
      <beneficiary>Bob</>
      <amount>5000</>
    </TransferFunds>
  </Body>
</Envelope>
```

Attacker has intercepted the message

This reference is not valid anymore because No of header is not 2. After attack it is 3

Attacker has added a BogusHeader & included the Body

Amount has been changed to 5000 by the attacker

Fig. 10. SOAP request after an attempt to attack (Excerpt)

We define:

- Number of children of Envelope is 2
- Number of Header is 2.
- Number of Signed Elements is 3

However:

- Vulnerable to Replay Attack itself (i.e. forgery SOAPAccount header)

Defense: WS-SecurityPolicy

```
<sp:SignedParts xmlns:sp="http://...securitypolicy">
  <sp:Body/>
  <sp:Header Name="To"
    Namespace="http://.../ws/2004/08/addressing"/>
  <sp:Header Name="From"
    Namespace="http://.../ws/2004/08/addressing"/>
</sp:SignedParts>

<sp:EncryptedParts xmlns:sp="http://...securitypolicy">
  <sp:Body/>
</sp:EncryptedParts>
```

```
<wsp:Policy wsu:Id="WSS10Anonymous with Certificates input policy">
  <wsp:ExactlyOne>
    <wsp:All>
      <sp:SignedParts>
        <sp:Body/>
      </sp:SignedParts>
      <sp:EncryptedParts>
        <sp:Body/>
      </sp:EncryptedParts>
    </wsp:All>
  </wsp:ExactlyOne>
</wsp:Policy>
```

- Domain specific language based on WS-Policy, expressed in WSDL
- Define things to be protected
- Define token to be used
- Define cryptographic communications protocols
- Uses low-level mechanisms that build and check individual security headers.
- **Too complicated. Hard to get right**

Our Goals

We want to define abstract and application level goals and turn it into WS-SecurityPolicy configuration files.

1. Easy to Write/Update [Less Painful]

Link language is a simple notation, covering some common cases, and could easily be generated from a simple UI or a systems modelling tool.

2. Security

It is safer to generate policy files from link specifications than write them directly.

Verify policy files

Architecture of Policy Files

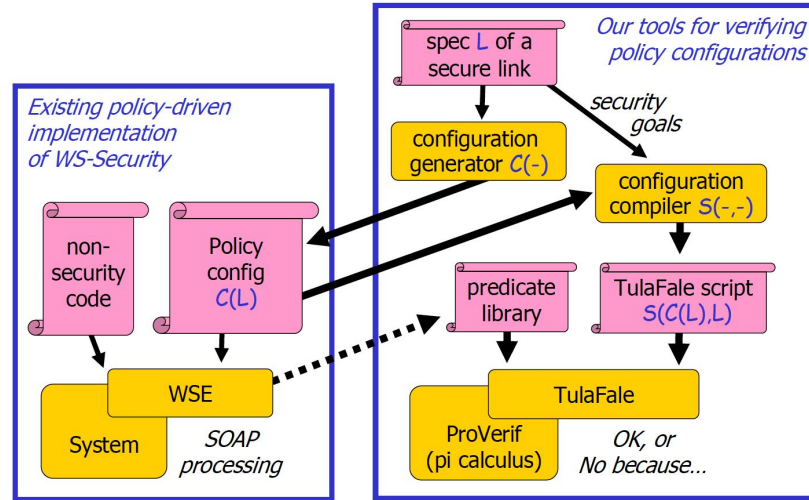


Figure 1: Generating and Checking Web Services Security Policies

TulaFale

pi-Calculus

ProVerif

Links -> Policies

Policies -> Predicate

Authentication and Adequacy Goals

A process P is ***robustly safe*** when, for any run in any context, if end $\text{Log}(a)$ occurs, then either begin $\text{Log}(a)$ or begin $\text{Leak}(u)$ with $[a = u @]$ previously occurred.

P is ***functionally adequate for a*** when, for some run in some context, end $\text{Log}(a)$ occurs.

```
process GenericSender() =  
  !in initChan(env);  
  in dbChan(sid); in dbChan(rid);  
  new freshid;  
  filter mkConformant(env,[sid],[rid],[freshid],outenv)  
→ !outenv in  
  filter linkAssert(sid,rid,env,a) → a in begin (Log,a);  
  out (httpChan, outenv)
```

```
process GenericReceiver() =  
  !in httpChan(env);  
  in dbChan(sid); in dbChan(rid);  
  filter isConformant(env,[sid],[rid],outenv) → outenv in  
  filter linkAssert(sid,rid,outenv,a) → a in end (Log,a);  
  done
```

Useful Predicates

```
predicate mkConformant(env:item,sids, rids, fresh:items,outenv:item) :-  
    hasHeaderTo(env,Toitm,Toval),  
    hasHeaderAction(env,Actionitm,Actionval),  
    Toval = "http://bobspetshop.com/service.asmx",  
    Actionval = "http://petshop/regular",  
    hasSendPolicyClientToService(env,sids,rids,fresh,outenv).
```

```
predicate hasReceivePolicyServiceToClient(env:item, sids, rids:items,  
                                         outenv:item) :-  
    hasResponseParts(env,Fromitm,RelatesToitm,MessageIditm,  
                     CreatedItm,Bodyitm),  
    hasSecurityHeader(env,toks),  
    xtok in toks, sig in toks,  
    isX509Token(xtok,"BobsPetShop",k,sids),  
    isSignature(sig,"rsasha1",k,  
               [Fromitm,RelatesToitm,MessageIditm,Createditm,  
                Bodyitm]),  
    outenv = env.
```

```
predicate isConformant(env:item, sids, rids:items, outenv:item) :-  
    hasReceivePolicyServiceToClient(env,sids,rids,outenv).
```

```
predicate linkAssert(sid, rid:item, env:item, a:items) :-  
    hasUid(sid,sender), hasUid(rid,responder)  
    hasHeaderTo(env,Toitm,to),  
    hasHeaderAction(env,Actionitm,action),  
    hasHeaderMessageId(env,MessageIditm,id),  
    hasHeaderCreated(env,Createditm,t),  
    hasBody(env,bitm,body),  
    to = "http://bobspetshop.com/service.asmx",  
    action = "http://premium",  
    responder in ["BobsPetshop"],  
    a = [sender responder "Request" to action id timestamp body].
```

```
predicate hasSendPolicyClientToService(env:item, sids, rids, fresh:items,  
                                       outenv:item) :-  
    sids = [user @ _],  
    isUserPassword(user,u,p),  
    fresh = [NewMessageId n t @ _],  
    hasRequestParts(env,Toitm,Actionitm,MessageIditm,  
                   CreatedItm,Bodyitm),  
    MessageIditm = <MessageId>NewMessageId</>,  
    mkUserTokenKey(utok,u,p,n,t,k),  
    mkSignature(sig,"hmacsha1",k,  
               [Toitm,Actionitm,MessageIditm,Createditm,Bodyitm]),  
    outenv = <Envelope>  
            <Header>  
                Toitm Acitm MessageIditm  
            <Security>  
                <Timestamp>Createditm</>  
                utok sig </></>  
            Bodyitm </>
```

Security Models: Secrecy

The generic sender inputs a request envelope from the attacker for recipient u

Try to replace the body by a secret name B

P preserves **secrecy** when, for any run in any context where B does not occur, if the context obtains B , then begin $\text{Leak}(u)$ and begin $\text{KnowsSecret}(u)$ previously occurred.

Security Models: Correlation

When the client accepts a response message from the web service, we want to guarantee that this message was generated in response to a particular earlier request.

```
process GenericClient() =  
  !in initChan (env);  
    in dbChan (cid); in dbChan (sid);  
    new freshid;  
    filter mkConformant(env,[cid],[sid],[freshid],outenv) → outenv in  
    filter linkAssert(cid,sid,env,aReq) → aReq in  
    begin Log(aReq);  
    out httpChan(outenv);  
  
    in httpChan(respenv);  
    filter isConformant(respenv,[sid],[cid],resp) → resp in  
    filter hasCorrelator(resp,freshid,cid) → in  
    filter hasLinkAssert(sid,cid,resp,aResp) → aResp in  
    end Log(aResp);  
    end LogCorr(aReq,aResp)
```

Discussion

We already have TLS (encryption) or other lower level mechanism to ensure integrity. Why do we still need this (Security-Policy)?