BitFlip: Determine a Data’s Signature Coverage from within the Application

Henrich C. Pöhls
Institute of IT-Security and Security Law
University of Passau, Germany
hp@sec.uni-passau.de
+498515093217

Copyright © The OWASP Foundation
Permission is granted to copy, distribute and/or modify this document under the terms of the OWASP License.

The OWASP Foundation
http://www.owasp.org
me, myself and this talk

- M.Sc. Information Security from Royal Holloway
- Diplom Informatik from University of Hamburg
- currently PhD student at University of Passau

ReSCUe IT:

- General: IT supported robust & secure Supply Chains
- Our Goal: Legally compliant & manageable integrity and authenticity statements for the data
Outline

- Problem & Motivation
- BitFlip Approach
- What BitFlip is not ...
- What BitFlip can do ... example XML-wrapping
- Conclusion
Problem: Message Security Layer vs. Application Layer

Design is layered

- Application Layer
  - Application logic works on data
- Data comes in by message
  - Application extracts data from message
- Security layer protects message (or part thereof)
  - Signed messages are verified before given to app.

Layered Security:
interlinking between layers must stay “in-sync”
Example: XML SOAP message security

Available security mechanism for SOAP messages:
- WS-Security (Tokens ... )
- XML Signature (and Encryption)

Security checks considered “good practice”:
- well defined XML schema
- rigorous schema validation
- validity check of signing public-key
- enforce strict security policies
Example: XML SOAP message security

Available security mechanism for SOAP messages:
- WS-Security (Tokens ... )
- XML Signature (and Encryption)

Security checks considered “good practice”:  
- well defined XML schema
- rigorous schema validation
- validity check of signing public-key
- enforce strict security policies

... but attacks on real world web services happen.
Authenticate not just the message, but everything that is used to determine the meaning of the message.

Ferguson and Schneier
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
  <soap:Header>
  </soap:Header>
  <soap:Body Id="1">
    <nds:return_hash> <!-- Optional: -->
      <nds:name>?fffd g</nds:name>
    </nds:return_hash>
  </soap:Body>
</soap:Body>
</soap:Header>
</soap:Body>

<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">  
  <SignedInfo>
    <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
    <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
    <Reference URI="#xpointer(id('1'))">  
      <Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
    </Reference>
  </SignedInfo>
</Signature>
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<soap:Header>
<wsse:Security xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">...
</soap:Body>
</soap:Header>
</soap:Envelope>

<Reference URI="#xpointer(id('1'))"/>
<Transforms>
<Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
</Transforms>
<SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
</Signature>
</soap:Body>
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope" xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
<soap:Header>
<wsse:Security xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
<soap:Body Id="1">
<nds:return_hash> <!--Optional:--> </nds:return_hash>
<nds:name>?fff d g</nds:name>
</nds:return_hash>
</soap:Body>
</soap:Header>
</soap:Body>
</soap:Envelope>
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
<SignedInfo>
<CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
<SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
<Reference URI="#xpointer(id('1'))"></Reference>
<Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
</SignedInfo>
</Signature>
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope" xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
<soap:Header>
<wsse:Security xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
   <soap:Body Id="1">
      <nds:return_hash> <!--Optional:--> 
         <nds:name>fff</nds:name>
      </nds:return_hash>
   </soap:Body>
</soap:Header>
<soap:Body>
   <nds:return_hash> <!--Optional:--> 
      <nds:name>evilHomer</nds:name>
   </nds:return_hash>
</soap:Body>
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#"><SignedInfo>
<CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
<SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
<Reference URI="#xpointer(id('1'))"> <!--Optional:-->
<Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
</Reference>
</SignedInfo>
</Signature>
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope" xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200408-wss-wssecurity-secext-1.0.xsd">
<soap:Header>
<wsse:Security xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200408-wss-wssecurity-secext-1.0.xsd">...
</soap:Header>
<soap:Body Id="1">
<nds:return_hash> <!--Optional:-->
<nds:name>?fff d g</nds:name>
</nds:return_hash>
</soap:Body>
</soap:Header>
<soap:Body>
<nds:return_hash> <!--Optional:-->
<nds:name>evilHomer</nds:name>
</nds:return_hash>
</soap:Body>
</soap:Envelope>

SOAP message example is from:

Meiko Jensen
Ruhr Universität Bochum
BitFlip: Observing the Signature Verification Outcome on Application Induced Errors

Signature Verification

VERIFIED
BitFlip: Observing the Signature Verification Outcome on Application Induced Errors

Signature Verification

VERIFIED
BitFlip: Observing the Signature Verification Outcome on Application Induced Errors

BitFlip: controlled change of single character
BitFlip: Observing the Signature Verification Outcome on Application Induced Errors

BitFlip: controlled change of single character

Signature Verification

VERIFIED

Signature Verification
BitFlip: Observing the Signature Verification Outcome on Application Induced Errors

Signature Verification

VERIFIED

BitFlip: controlled change of single character

Signature Verification

VERIFIED
BitFlip: Observing the Signature Verification Outcome on Application Induced Errors

BitFlip: controlled change of single character

Signature Verification

VERIFIED ✔

Signature Verification

VERIFIED ✗
BitFlip: Observing the Signature Verification Outcome on Application Induced Errors

Signature Verification

VERIFIED

Result: character not covered

Signature Verification

VERIFIED

BitFlip: controlled change of single character

Henrich C. Pöhls

Institute of IT-Security and Security Law

OWASP
BitFlip: Observing the Signature Verification Outcome on Application Induced Errors

BitFlip: controlled change of single character

Signature Verification

VERIFIED

✓

Signature Verification

SUCCESS

✓

Signature Verification

FAILURE

✗
BitFlip: Observing the Signature Verification Outcome on Application Induced Errors

Signature Verification

VERIFIED ✅

Signature Verification

FAILURE ❌

BitFlip: controlled change of single character

Result: character covered

Return_hash> <!--Option name>?fffdf g/nnt$s:nam>

Signature Verification

VERIFIED ✅

Signature Verification

FAILURE ❌
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope" xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
<soap:Header>
<wsse:Security xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
<soap:Body Id="1">
<nds:return_hash> <!--Optional:--> 
<nds:name>?fff dg</nds:name>
</nds:return_hash>
</soap:Body>
</soap:Header>
</soap:Body>
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#"><SignedInfo>
<CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
<SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
<Reference URI="#xpointer(id('1'))"><Transforms>
<Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
</Reference>
</SignedInfo>
</Signature>
  <soap:Header>  
      <soap:Body Id="1">  
        <nd5:return_hash> <!--Optional:-->  
          <nd5:name>?fffdf g</nd5:name>  
        </nd5:return_hash>  
      </soap:Body>  
    </wsse:Security>  
  </soap:Header>  
  <soap:Body>  
    <nd5:return_hash> <!--Optional:-->  
      <nd5:name>evilHomer</nd5:name>  
    </nd5:return_hash>  
  </soap:Body>  
  <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">  
    <SignedInfo>  
      <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315">  
        <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>  
        <Reference URI="#xpointer(id('1'))"/>  
        <Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>  
      </CanonicalizationMethod>  
      <SignatureValue>  
        <SignedInfo>  
          <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>  
          <Reference URI="#xpointer(id('1'))"/>  
          <Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>  
        </SignatureValue></Reference>  
      </Reference>  
    </SignedInfo>  
    <SignatureValue>  
      <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>  
      <Reference URI="#xpointer(id('1'))"/>  
      <Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>  
    </SignatureValue></Reference>  
  </Signature>  
</soap:Envelope>
Not covered by Signature
Covered by Signature

XML file is corrupt
Signature couldn't be found

Internals from JAVA verify process:
java.xml.crypto.dsig.XMLSignature

Marshal Exception
Nullpointer while unmarshaling
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope"

<soap:Header>
<wsse:Security xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
</soap:Body>
<soap:Body>
<nds:return_hash> <!--Optional:--> <nds:name>?fffd g</nds:name>
</nds:return_hash>
</soap:Body>
<soap:Body>
<nds:return_hash> <!--Optional:--> <nds:name>evilHomer</nds:name>
</nds:return_hash>
</soap:Body>
</soap:Body>
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#"> <SignedInfo>
<CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
<SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
<Reference URI="#xpointer(id('1'))"> <Transforms>
<Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
</Transform>
</Reference>
</SignedInfo>
</Signature>
What BitFlip does NOT and can NOT offer ...

BitFlip does not do “positive verification”:

- no assurance that the parts that seem covered are secured against all kinds of attacks
- does not check for exploits in the signature verification process
- is not a “fuzzer”
What BitFlip does ...

- detects absence of integrity protection ("white spots")
- works independently of signature verification process ("black-box")
- implemented on application level
  - application controlled
  - use same "parser logic" to select flipping data
- absence can be detected by a single "flip"
  - overhead of one additional signature verification
BitFlip: Conclusion

- Allows Applications to test if Signature Verification Process covers the data the application logic extracted
- Independent of Verification Process (black-box)
  - Full Verification not necessary if no black-box
- Tool to evaluate the Verification Process
  - detect errors during application design
  - testing the layers below before application roll-out
  - re-run tests after changes to the policy or the verification process