Verifying Web Services Security

Karthikeyan Bhargavan

Joint work with C. Fournet & A.D. Gordon
with contributions from R. Pucella and R. Corin

MSR Cambridge Samoa Project
http://securing.ws

Securing Web Services?

- Web Services are a new way of building distributed systems
  - XML/SOAP messages sent over HTTP/SMTP
  - + Multi-party workflow + Service Discovery + Policy Exchange + Security
- Web Services are the next big thing for distributed messaging
  - At least Microsoft, IBM, Sun, W3C, … think so
  - Toolkits available now: Microsoft WSE, Sun WSTK, IBM, Verisign
  - Lack of security assurances is biggest roadblock (Gartner, 2003)
  - New specifications are being designed for securing web services
  - Security is notoriously hard, XML or no XML
- Meanwhile in research-land…
  Sustained and successful effort to develop formalisms and tools to check crypto protocols
  - Needham-Schroeder threat model: attacker can capture, replay, redirect, rewrite messages, but cannot guess secrets or break crypto
  - Hot Research Topic: approx 30 papers per year
- Timely opportunity to develop tools for validating standards-based XML crypto protocols

MSR Cambridge Samoa Project

- Verify strong security properties for web services deployments
  - Theorems on Secrecy, Integrity, Authentication
  - Published papers in POPL'03, FMCO'03, CCS'04, SWS'04
- Give a formal model for new protocol specifications
  - Suggested improvements
- Validate our model against sample configurations distributed with WSE (MB)
  - Found numerous attacks, helped fix them, took part in security reviews
  - Proved samples correct
- Develop automated analysis tools
  - Release TulaFale, PolicyAdvisor
- This talk:
  - TulaFale: our language for specifying web services security protocols
  - Policy Analyzer: automatically verify system configurations relying on web services
  - Policy Generator: generate secure-by-default security configurations
  - Policy Advisor: suggest best practices to developers for their security policies

Talk Outline

1. Intro to Web Services Security and Threats
2. Discussion of protocol vulnerabilities
3. TulaFale: Verifying protocols
   - Tool for protocol designers, researchers, spec authors
4. Policy Analyzer: Verifying system configs
5. Policy Generator: Generating secure configs
   - Tools for web services developers, administrators
6. Policy Advisor: Suggesting best practices
   - Tool for end users, security non-experts
7. Conclusions and Ongoing work

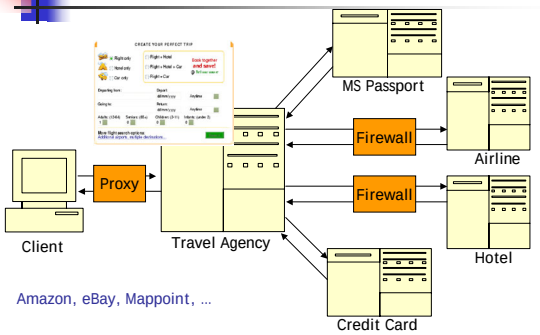
Part I: Web Services & Security

Web Sites vs. Web Services

Every evening:
- MS Money queries Bank's Web Service with Username/Password
- Bank sends statement

1. Browse to Bank's Web Site
2. Enter Username/Password
3. Download Statement
4. Import into MS Money

The Future of E-Commerce?

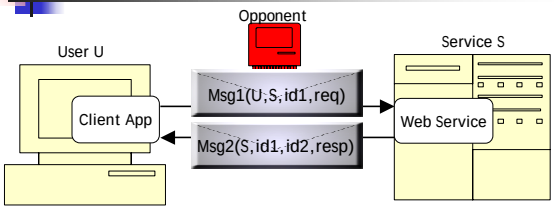


Amazon, eBay, Mappoint, …

Security Threats

- Applications available over open web
  - Traditional web application attacks
    - Input Validation
    - Denial of Service
- New concerns: flexible, XML-based protocols
  - Web services developers can design and deploy their own security protocols
    - Not just a fixed protocol like SSL
    - Can combine standard protocols in new ways
  - XML message format open to rewriting attacks
    - Much like classic active attacks (Needham-Schroeder '78)
    - Opponent can Redirect, Replay, Modify, Impersonate
    - New: flexible, semi-structured message formats
      - structure of received message triggers processing

Simple RPC Protocol

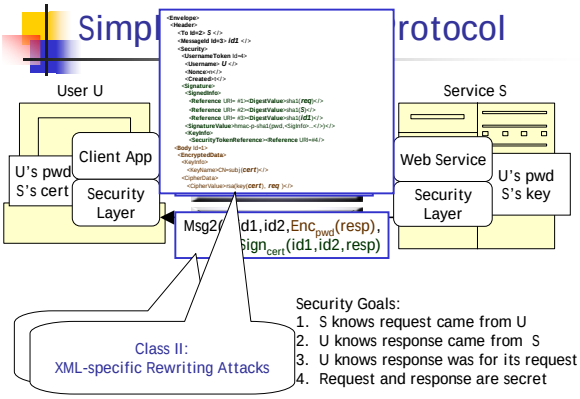


If Msg1 and Msg2 are in the clear, the opponent can
- Read U's request and response
- Change Msg1 to (U,T,id1,req) to redirect message to different service T
- Change Msg2 to (S,id1,id2,resp') to fool U about her response
- …

Available Security Mechanisms

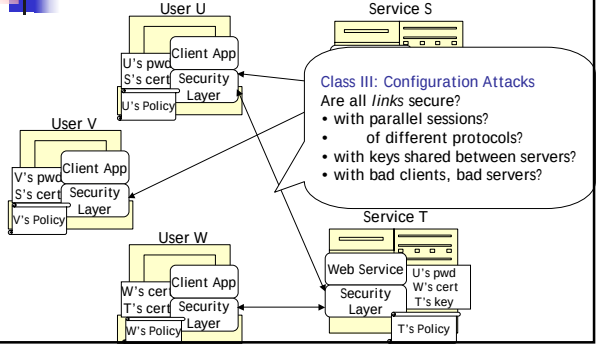
- Series of emerging specifications for
  - message crypto (WS-Security)
    - Digital signatures, Encryption of message parts
  - security token exchange (WS-Trust)
  - secure sessions (WS-SecureConversation)
  - security policies (WS-SecurityPolicy)
  - … and others, from Microsoft, IBM, Verisign, …
- Clients and servers can secure exchanges using protocols defined in these specs
  - deploying security policy files that choose, combine, and configure protocols

Simpl rotocol



Security Goals:
1. S knows request came from U
2. U knows response came from S
3. U knows response was for its request
4. Request and response are secret

Multiple Clients and Servers

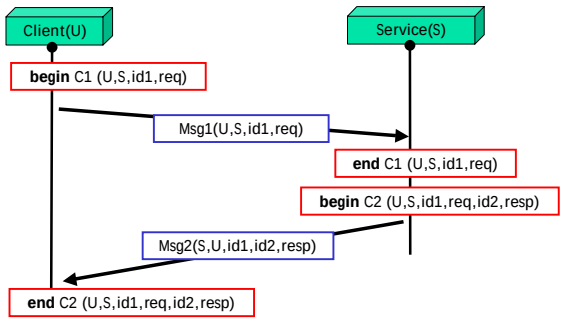


Summary: Threat Model

- The attacker
  - Controls the network
    - Chooses an arbitrary network topology
    - Sees all messages
    - Controls message routing and delivery
  - Controls several (bad) clients, services, token providers
    - Knows all public keys for good principals
  - Can initiate any number of sessions, message exchanges
  - Can open and modify messages
    - using known XML formats and previously obtained keys
- But, the attacker cannot
  - Act as Root CA for X.509 certificates
  - Guess secrets such as passwords or keys
  - Break cryptographic algorithms
- Goal: Message authentication, correlation, and confidentiality
  - for compliant (good) clients, services, token providers

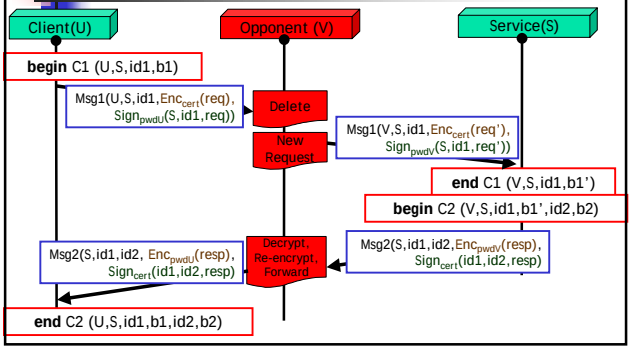
Part II: Protocol Vulnerabilities

An intended run of the sample protocol



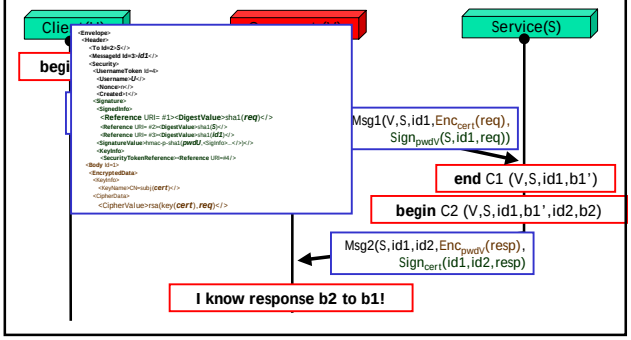
Security Goals:
1. Request Authentication: Client and Service agree on C1
2. Response Authentication and
   Request-Response Correlation: Client and Service agree on C2
3. Secrecy: Attacker cannot compute req and resp from messages

Attack I
- Opponent breaks correlation by forwarding his response to U
- Class I: Classic M-i-M protocol attack
- Possible Fixes: Sign U in Msg2, or Encrypt id1 in Msg1

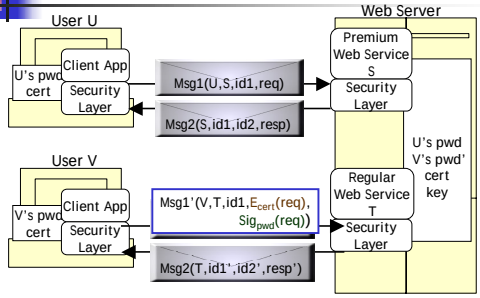


Attack II

- Opponent breaks response secrecy by re-signing U's request
- Class II: XML-specific rewriting attack
- Fix: Encrypt signature in Msg1



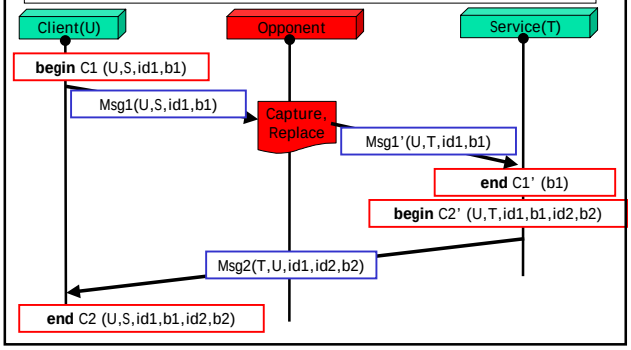
Multiple services with same signing key



Weak Protocol between V & T: Only req signed in Msg1'!
Security Goal: protocol between U & S must still be secure

Attack III

- Opponent fools U into accepting regular response for premium request
- Class III: Configuration-specific attack
- Fix: Sign S in Msg2 & Msg2', Don't share keys



Summary

- Found dozens of security flaws
- WSE 1.0, 2.0 modified
  - Participated in security reviews
- Some specs have been changed
- Best practices paper in the works
- Transfer of analysis tools in progress
  - http://securing.ws

Part III: TulaFale – Verifying Protocols

TulaFale

- Finding bugs is useful, but…
- we would like to give positive guarantees!
- TulaFale: New language for specifying XML-based security protocols
  - More abstract than C# code
    - focused only on security aspects
  - Small, formal language
    - good target for analysis tools
  - Based on pi calculus
    - easy to express, understand powerful attacker models
    - can use analysis tools such as Blanchet's ProVerif
  - Growing library of web service security protocols
    - easy to program new protocols, compose existing ones

The Pi-Calculus and Cryptography

The pi-calculus is a tiny yet highly expressive concurrent language, with precise semantics, rich theory, and several implementations

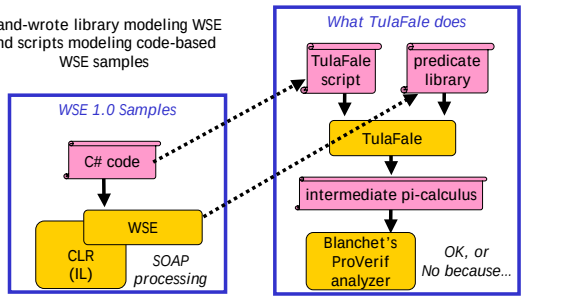
$P, Q ::=$	process
$\mathbf{out}\ x(y_1, \ldots, y_n)$	output
$\mathbf{in}\ x(z_1, \ldots, z_n);\ P$	input
$\mathbf{new}\ x;\ P$	fresh name
$P \mid Q$	parallel
$!P$	replication
$\mathbf{0}$	inactivity

- Milner, Parrow, Walker (1989); Milner (1999)
- The spi-calculus (Abadi and Gordon 1999) adds Dolev-Yao style representation of cryptographic operations and protocols
- Various proof techniques developed, including resolution-based prover ProVerif (Blanchet 2001), for reasoning about systems P | O where P is the explicit system and O is an arbitrary, unknown, active attacker

Analyzing Protocols using TulaFale

TulaFale = pi + XML + predicates + assertions

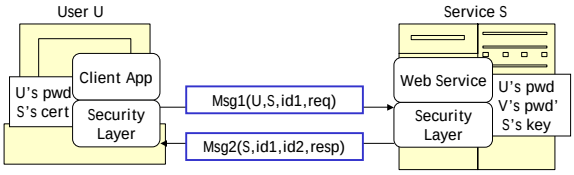
Hand-wrote library modeling WSE and scripts modeling code-based WSE samples



Modeling Protocols in TulaFale

- Protocol = principals + messages + processing + security goals
- TulaFale = pi + XML + predicates + assertions
- Messages modeled by XML terms with cryptography
  - terms in a many-sorted algebra with sorts for strings, XML elements, bytestrings, lists
  - include constructors/destructors modeling cryptography
- Message processing modeled by predicates
  - horn clauses over XML terms with message equalities
- Compliant (good) principals modeled by pi processes
- Non-compliant or compromised (bad) principals modeled by unspecified environment
  - "attacker can do anything, using the secrets it knows"
- Security goals modeled by assertions on events executed by good principals

Specifying Secure RPC Protocol



Specifying Protocols: Message Processing as Predicates

```
predicate isMsg1(msg1:item,U:item,pwd:string,S:item,skS:bytes,
                 id1:string,req:item) :-
  msg1 =
    <Envelope>
      <Header>
        <To>S</>
        <MessageId>id1</>
        <Security>
          utok
          sig1</></>
      <Body>b1</></>,
  isEncryptedData(b1,req,skS),
  isUserTokenKey(utok,U,pwd,skU),
  isSignature(sig1,"hmacsha1",skU,
              [<Body>req</> <To>S</> MessageId>id1</>]).
```

Parse XML Message

Decrypt Message Body (req)

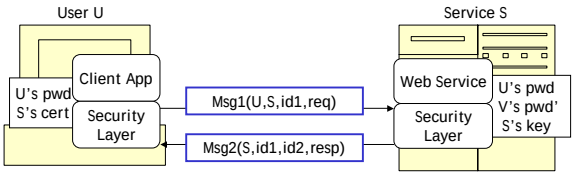
Compute pwd-based key

Check Signature

Uses library predicates for:
1. message parsing,
2. encryption, decryption
3. signatures,
4. tokens, key computation

covering WS-Security, …

Specifying Protocols: Clients, Services as Pi Processes



Specifying Protocols: Clients, Services as Pi Processes

```
process Client(U:string,pwd:string,pkR:bytes) =
  in init (S,certS,req,n,t);
  new id1:string;

  begin C1 (U,S,id1,req);
  filter mkMsg1(msg1,U,pwd,n,t,S,certS,id1,req) -> msg1;
  out soap(msg1);

  in soap(msg2);
  filter isMsg2(msg2,S,certS,id1,id2,resp) -> id2,resp;
  end C2 (U,S,id1,req,id2,resp).
```

Call predicate to construct Msg1 (send policy)

Message Exchange

Call predicate to check Msg2 (receive policy)

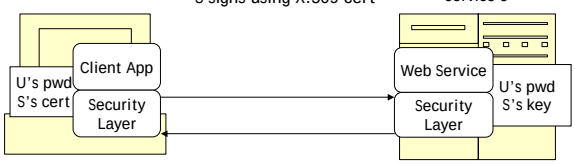
```
process Service(S:string,certS:bytes,skS:bytes) =
  in soap(msg1);
  in anyUser(U,pwd);
  filter isMsg1(msg1,U,pwd,S,skS,id1,req) -> id1,req;
  end C1 (U,S,id1,req);

  in accept (id2,resp);
  filter mkMsg2(msg2,S,certS,skS,id1,id2,resp) -> msg2;
  begin C2 (U,S,id1,req,id2,resp);
  out soap(msg2).
```

Automated Proofs

- Security Goals written as Queries for ProVerif
  - Request Authentication
    - For all honest U, S: whenever a Service(S,..) invokes end C1 (U,S,id1,req), some Client(U,..) must have invoked begin C1 (U,S,id1,req).
  - Response Authentication & Request-Response Correlation
  - Secrecy of req and resp
- For sample protocol
  - Request-Response Correlation, Response Secrecy does not hold
  - Proverif finds attacks I and II
- Theorem: All runs of (fixed) script preserve authentication, correlation and secrecy
  - for any number of client-service sessions and
  - for an attacker that can send, read, modify, replay any message
  - for any number of other clients, services in parallel
- Typical script size: 200 lines + library
  - Analysis time: few seconds to few minutes

TulaFale Demo

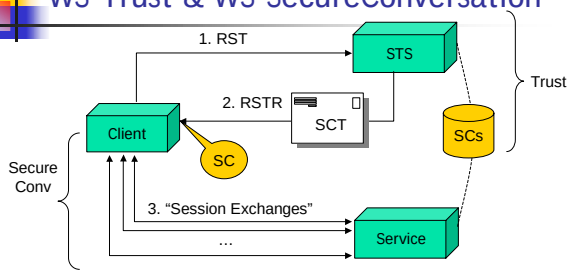
Based on WSE 2.0 Samples:
U signs using shared pwd
S signs using X.509 cert



Multi-Party Protocols

- WS-Security provides basic mechanisms to secure SOAP traffic, one message at a time
  - Signing and encryption keys derived from long-lived secrets like passwords or private keys
- If a SOAP interaction consists of multiple, related messages, WS-Security alone may be inefficient, and does not secure session integrity
  - Standard idea: establish short-lived session key
- Recent specs describe this idea at the SOAP-level
  - **WS-SecureConversation** defines *security contexts*, used to secure sessions between two parties
  - **WS-Trust** defines how security contexts are issued and obtained

Multi-Party Protocols: WS-Trust & WS-SecureConversation



STS = Security Token Server
RST = Request Security Token
RSTR = RST Response
SC = Security Context
SCT = SC Token

Discussion

- First formal analysis of WS-Trust and WS-SecureConversation
  - XML syntax and automation very effective, against a demanding, realistic attacker model
  - Approx 1000 LOC - manual proofs we published at POPL'04 concerning one or two message protocols would not scale
  - Still, a theorem concerning open-ended sessions proved by combination of automated proof and short hand-proof
- As is common, these specs:
  - focus on message formats for interoperability
  - are non-committal regarding security, for example, no clear spec of contents of SCs
- By making modes, data, and goals explicit, we found design and implementation bugs
  - see our paper at ACM SWS workshop for a discussion

Status and Ongoing work

- Models for several protocol standards
  - Covering most WSE 1.0, 2.0 samples
    - Username-passwords, X.509, SCT sessions, SAML
  - Some interop scenarios
- Library predicates for important specs
  - WS-Security, WS-Trust, WS-SecureConversation
- Published papers on case studies
- TulaFale 0.1 available for download
  - With library and models included
  - http://securing.ws

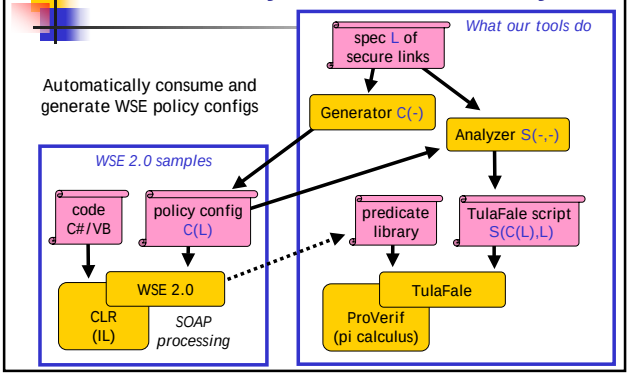
Part IV: Policy Analyzer – Verifying Configurations

Web Service Security Policies

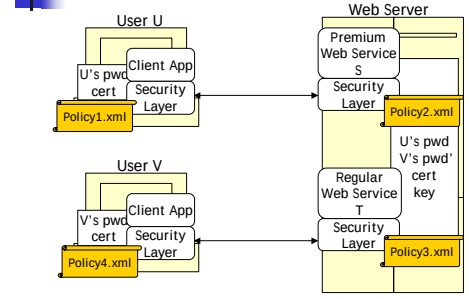
- Clients, services have XML files describing security policy

```
<Policy Id="Msg1">
  <All>
    <Confidentiality>
      <TokenInfo>
        <SecurityToken>
          <TokenType>X509v3</>
          <Claims><SubjectName>S</></>
      <MessageParts>Body()</>
    <Integrity>
      <TokenInfo>
        <SecurityToken>
          <TokenType>UsernameToken</>
          <Claims><SubjectName>U</></>
      <MessageParts>Body() Header("To")
                    Header("MessageId")</>
```

Tools: Policy Generator/Analyzer



Analyzing Policy Configurations



Automated tools for collecting, parsing policies from IIS Servers, Clients

Config = [Policy1, Policy2, Policy3, Policy4]

Translating Policies to Predicates

```
<Policy Id="Msg1">
  <All>
    <Confidentiality>
      <TokenInfo>
        <SecurityToken>
          <TokenType>X509v3</>
          <Claims><SubjectName>S</></>
      <MessageParts>Body()</>
    <Integrity>
      <TokenInfo>
        <SecurityToken>
          <TokenType>UsernameToken</>
          <Claims><SubjectName>U</></>
      <MessageParts>Body() Header("To")
                    Header("MessageId" )</>
```

Conjunction

Encryption Requirement

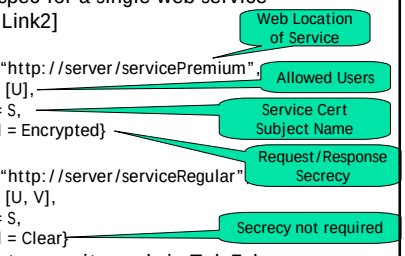
Signature Requirement

predicate hasMsg1Policy(msg1:item,U:item,pwd:string, S:item,skS:bytes, id1:string,req:item) :-

```
msg1 =
  <Envelope>
    <Header>
      <To>S</>
      <MessageId>id1</>
      <Security>
        utok
        sig1</></>
    <Body>b1</></>,
isEncryptedData(b1,req,skS),
isUserTokenKey(utok,U,pwd,skU),
isSignature(sig1,"hmacsha1",skU,
            [<Body>b1</> <To>S</> MessageId>id1</>]).
```

Link Specifications: Formalizing Desired Topology

- Link: Security spec for a single web service
- Spec = [Link1, Link2]
- Link1 =
  - {ServiceURI = "http://server/servicePremium",
  - ClientPrins = [U],
  - ServicePrin = S,
  - SecrecyLevel = Encrypted}
- Link2 =
  - {ServiceURI = "http://server/serviceRegular"
  - ClientPrins = [U, V],
  - ServicePrin = S,
  - SecrecyLevel = Clear}
- Links translate to security goals in TulaFale
  - All requests and responses on Link1 and Link2 must be secure



Proofs

- Policy Analyzer S(-,-) translates policies and links to TulaFale scripts
- For sample Config & Spec, assertions for both links fail
  - Proverif finds Attack III
  - If services use different certs, then Link1 is secure
    - but Link2 is not
  - If both services use strong policies, then both links are secure
- Similar results for other sample configurations
  - Sample policy: 69 lines,
  - Link spec: 14 lines,
  - Generated script: 283 lines (+library)
  - Analysis time: several minutes to several hours

Part V: Policy Generator: Secure-by-Default Configurations

Generating Secure Policies

- Policy generator C(-) translates links to secure-by-default policy configs
  - Makes specific choices on what to sign, encrypt
  - Specializes policy for desired links
- Generated policies can be directly deployed in WSE clients, servers
- Recommended policy stronger than defaults
  - In request, sign
    - <Body>, (Integrity)
    - <To>, <Action>, (Avoid Redirection)
    - <MessageId>, <Timestamp> (Avoid Replay)
  - In response, sign
    - <Body>, (Integrity)
    - <From>, <RelatesTo>, (Correlation)
      - <RequestingUser>?
    - <MessageId>, <TimeStamp> (Avoid Replay)
  - If encrypted link,
    - Encrypt <Body> in both request and response

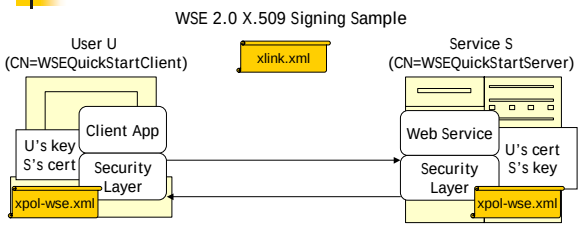
Generated Policies are Secure

- Theorem: All policy configurations generated from link specs satisfy our security goals
  - For all link specifications L, S(C(L),L) provides request & response authentication, correlation, and secrecy for links in L
- Hence, generated configs can be safely deployed
- Proof:
  - Strong property: there are an unlimited number of link specs
  - Use combination of automated proofs and manual reasoning
  - Hint 1: There are only 4 kinds of link generated policies
  - Hint 2: Prove that the config with all links enabled is secure

Service unaffected by Client Policies

- Theorem: If a service uses a link-generated policy, then irrespective of the client policies, the resulting configuration preserves request authentication and response secrecy
- Hence, naïve clients cannot break service security
- Proof:
  - Again, combination of automated proofs and manual reasoning
  - Hint: Even the weakest send policy preserves secrecy of passwords, signing keys

Policy Generator/Analyzer Demo



Part VI: Policy Advisor – Suggesting Best Practices

How Do Policies Fail?

- Policy-based security is a Good Thing compared to code-based security, but no panacea
- During security reviews of sample policies, we found several common errors
  - Replay attacks – failure to sign message ID and timestamp
  - Redirection attacks – failure to sign the <To> or <Action> headers
  - Dictionary attacks – unencrypted passwords, even if hashed
  - Man-in-the-middle attacks – failure to bind initiator identity to response messages
- Unlikely to be fixed once for all in a standard, since policies explicitly intended to be customisable
- Security-savvy developers could use Policy Analyzer to verify policy configs
- Need for a light-weight tool to quickly advise on best practices

Policy Advisor

- Loads a WSE2 policy config, runs queries, generates a report (errors, warnings, assurances)
- Offers only relative guarantees, but aims to help understanding of policies, and to catch typical errors
  - TulaFale part of testing process, but not part of tool
- Suggests best practices for policy configs
- Status: queries for password disclosure and replay, redirection, and dictionary attacks
  - Still quite rudimentary
  - Public release soon…

Part VII: Conclusions

Related Work

- Going in the opposite direction to our policy analyzer, several tools compile formal models to code:
  - Strand spaces: Perrig, Song, Phan (2001), Lukell et al (2003)
  - CAPSL: Muller and Millen (2001)
  - Spi calculus: Lashari (2002), Pozza, Sista, Durante (2004)
  - Apparently, the resulting code cannot yet interoperate with other implementations – an important future target
- Other Dolev-Yao modelling of web services
  - Type-based analysis of pre-WS-Security web services using Cryptyc: Gordon and Pucella (2002)
  - Model-checking of some example WS-Security specs using FDR, uncovering similar attacks: Kleiner & Roscoe (2004)
- Other formalizations of XML and web services specs
  - XPath, XSLT, XQuery: Wadler et al (since 1999)
  - WS-AT: Johnson, Langworthy, Lamport, Vogt (2004)

Summary & Conclusions

- The first automated analysis of security for a deployed system based on crypto protocols
  - Our tools find attacks, verify samples, generate secure policies
  - Analysis tools to be transferred to developers
- Web services security specs enable a lot of flexibility
  - Message formats, composable protocols, configurations
  - Makes guaranteeing security difficult!
- Specs and implementations are only just emerging
  - Notoriously difficult to get right
  - Attacks and proofs are subtle, need tool support
- Important to verify configurations
  - Proving protocols secure is not enough
- Good place to develop formal tools, get positive results
  - Standard message formats
  - Wide applicability

Ongoing and Future Work

- Many new protocols
  - Moving from WSE to Indigo
  - And new specifications: WS-Security over TLS
- Release tools
  - TulaFale for protocol analyses
  - Lightweight “best practices” advisor
- Extend link specifications and results
  - Token servers, Firewalls,
  - Multi-party exchanges,
  - Authorization

Questions?

http://securing.ws