Verifying Web Services Security Configurations

Karthikeyan Bhargavan

Samoa Project
http://securing.ws
Microsoft Research Cambridge

Abstract. XML Web Services provide a flexible API for building distributed systems as a collection of endpoints that can send and receive SOAP messages. These systems are secured using message-based cryptographic mechanisms defined in a series of specifications developed by Microsoft, IBM, and others. Such home-grown security protocols often go wrong; they are prone to a well-known class of attacks, formalized by Dolev and Yao, where an attacker can intercept, modify, and replay messages. The vulnerability is only increased by the flexible message formats and complex trust configurations allowed by the standards. Our goal is to verify the security of families of protocol configurations, such as those deployed for Microsoft's WSE and Indigo web services implementations.

We propose a new specification language for writing machine-checkable descriptions of SOAP-based security protocol configurations and their properties. Our TulaFale language is based on the pi calculus (for writing collections of SOAP processors running in parallel), plus XML syntax (to express SOAP messaging), logical predicates (to construct and filter SOAP messages), and correspondence assertions (to specify authentication goals of protocols). Our implementation compiles TulaFale into the applied pi calculus, and then runs Blanchet's resolution-based protocol verifier. The TulaFale implementation is available for download.

We also describe a high-level *link* language for describing security configurations, and demonstrate a tool that can automatically generate and analyze executable security policy deployments for web services written using WSE. This is the first tool we know of that can automatically analyze cryptographic configurations to find real errors and demonstrable attacks.

1 Introduction

Web services enable a server to provide authorized users with programmatic access to data and software over the web. For instance, an e-commerce web site may distribute to its customers client software that regularly connects to the site and downloads the latest stock and price information (in XML) to update a local database. This level of access exposes the server to increased security risks; hence, our goal is to verify the security of such web service deployments.

This work was done in collaboration with A.D. Gordon, C. Fournet, R. Corin, and R. Pucella

The security configuration for such a server includes all its exported web services, the corresponding clients, the customer database, and the security policies and application-level security at each client and service. Various emerging web services security specifications define standard mechanisms for securing such (and more complex) configurations; and emerging APIs such as Microsoft's WSE enable developers to use these standard mechanisms to secure their deployed web services.

In the rest of this section, we present a brief introduction to web services security specifications, providing additional references for the interested reader. In the next section, we describe our approach toward the automated verification of web services configurations; we outline our tools, referring to published papers for details.

1.1 Web Services

Web services are a wide-area distributed systems technology, based on asynchronous exchanges of XML messages conforming to the SOAP message format [BEK+00,W3C03]. A basic motivation for web services is to support programmatic access to web data. The HTML returned by a typical website is a mixture of data and presentational markup, well suited for human browsing, but the presence of markup makes HTML a messy and brittle format for data processing. In contrast, the XML returned by a web service is just the data, with some clearly distinguished metadata, well suited for programmatic access. For example, search engines export web services for programmatic web search, and e-commerce sites export web services to allow affiliated websites direct access to their databases.

Generally, a broad range of applications for web services is emerging, from the well-established use of SOAP as a platform and vendor neutral middleware within a single organisation, to the proposed use of SOAP for device-to-device interaction [S+04]. Still, the simplest and predominant application of web services is to implement remote procedure calls over the web (HTTP) with arguments and results serialized as XML. (However, RPCs are not their only application [Vog03].)

1.2 Securing Web Services with Cryptographic Protocols

Web services specifications support SOAP-level security via a syntax for embedding cryptographic materials in SOAP messages. The WS-Security standard [NKHBM04] describes how to sign and encrypt portions of SOAP messages, so as to achieve end-to-end security. Hence, to meet their security goals, web services and their clients can construct and check *security headers* in messages, according to the WS-Security format [IM02,NKHBM04]. Related specifications such as WS-Trust [KN+04b] and WS-SecureConversation [KN+04a] extend WS-Security with key-exchange mechanisms and session-level security.

WS-Security can provide message confidentiality and authentication independently of the underlying transport, using, for instance, secure hash functions,

shared-key encryption, or public-key cryptography. WS-Security has several advantages compared to using a secure transport such as SSL, including scalability, flexibility, transparency to intermediaries such as firewalls, and support for non-repudiation. Significantly, though, WS-Security does not itself prescribe a particular security protocol: each application must determine its security goals, and process security headers accordingly.

Web services may be vulnerable to many of the well-documented classes of attack on ordinary websites [SS02,HL03]. Moreover, unlike typical websites, web services relying on SOAP-based cryptographic protocols may additionally be vulnerable to a new class of *XML rewriting attacks*: a range of attacks in which an attacker may record, modify, replay, and redirect SOAP messages, but without breaking the underlying cryptographic algorithms. Flexibility comes at a price in terms of security, and it is surprisingly easy to misinterpret the guarantees actually obtained from processing security headers. XML is hence a new setting for an old problem going back to Needham and Schroeder's pioneering work on authentication protocols; SOAP security protocols should be judged safe, or not, with respect to an attacker who is able to "interpose a computer on all communication paths, and thus can alter or copy parts of messages, replay messages, or emit false material" [NS78]. XML rewriting attacks are included in the WS–I threat model [DHK+04]. We have found a variety of replay and impersonation attacks in practice.

1.3 Configuring Web Services Security with Policy

Rather than using APIs for manipulating security tokens, application writers are encouraged to state their security requirements in configuration files. WS-SecurityPolicy [DLHBH+02], built on the WS-Policy [BCH+03] and WS-PolicyAssertion [BHK+03], is a declarative XML format for programming how web services implementations construct and check WS-Security headers. By expressing security checks as XML metadata instead of imperative code, policy-based web services conform to the general principle, when building secure systems, of isolating security checks from other aspects of message processing, to aid human review of security. Moreover, coding security checks as XML metadata aids interoperability since the metadata may easily be exchanged between different implementations on different platforms.

Still, driving web services security from WS-SecurityPolicy is no panacea. First, despite its name, WS-SecurityPolicy drives low-level mechanisms that build and check individual security headers; we need a way to relate policies to more abstract, application-level goals such as message authentication or secrecy. Second, the configuration files, including WS-SecurityPolicy files, of a SOAP-based system largely determine its vulnerability to XML rewriting attacks; WS-SecurityPolicy gives freedom to invent new cryptographic protocols, which are hard to get right, in whatever guise.

2 Verifying Web Services Security

The use of formal methods to analyze traditional cryptographic protocols and their vulnerabilities began with work by Dolev and Yao [DY83]. In the past few years there has been intense research on the Dolev-Yao model, leading to the development of numerous formalisms and tools.

Our work builds on the line of research using the pi calculus. The pi calculus [Mil99] is a general theory of interaction between concurrent processes. Several variants of the pi calculus, including spi [AG99], and a generalization, applied pi [AF01], have been used to formalize and prove properties of cryptographic protocols. A range of compositional reasoning techniques is available for proving protocol properties, but proofs typically require human skill and determination. Recently, however, Blanchet [Bla01,Bla02] has proposed a range of automatic techniques, embodied in his theorem prover ProVerif, for checking certain secrecy and authentication properties of the applied pi calculus. ProVerif works by compiling the pi calculus to Horn clauses and then running resolution-based algorithms.

2.1 TulaFale: A Security Tool for Web Services

TulaFale is a new scripting language for specifying SOAP security protocols, and verifying the absence of XML rewriting attacks:

$$\text{TulaFale} = \text{processes} + \text{XML} + \text{predicates} + \text{assertions}$$

The pi calculus is the core of TulaFale, and allows us to describe SOAP processors, such as clients and servers, as communicating processes. We extend the pi calculus with a syntax for XML plus symbolic cryptographic operations; hence, we can directly express SOAP messaging with WS-Security headers. We declaratively specify the construction and checking of SOAP messages using Prolog-style predicates; hence, we can describe the operational details of SOAP processing. Independently, we specify security goals using various assertions, such as correspondences for message authentication and correlation.

It is important that TulaFale can express the detailed structure of XML signatures and encryption so as to catch low-level attacks on this structure, such as copying part of an XML signature into another; more abstract representations of message formats, typical in the study of the Dolev-Yao model and used for instance in previous work on SOAP authentication protocols [GP03], are insensitive to such attacks.

Our methodology when developing TulaFale has been to study particular web services implementations, and to develop TulaFale scripts modelling their security aspects. Our experiments have been based on the WSE development kit [Mic02], a particular implementation of WS-Security and related specifications. We have implemented the running example protocol of this paper using WSE, and checked that the SOAP messages specified in our script faithfully reflect the SOAP messages observed in this implementation. For a discussion of

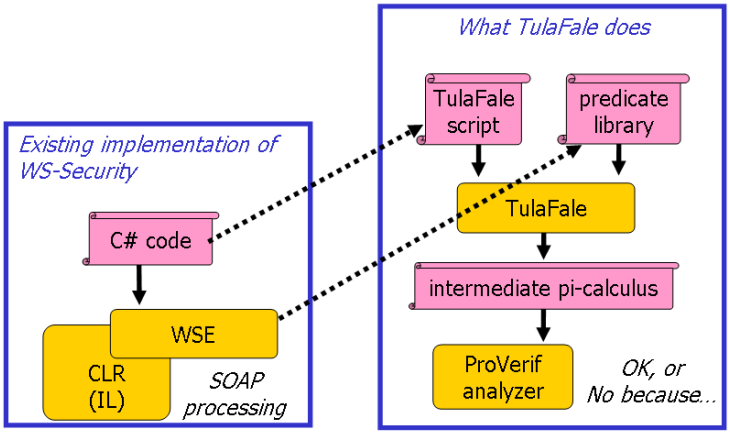

Fig. 1. Modelling WS-Security protocols with TulaFale

the implementation of related protocols, including logs of SOAP messages, see the technical report version of an earlier paper [BFG04a].

Fig. 1 illustrates our methodology. On the left, we have the user-supplied code for implementing a web services protocol, such as the one of this paper, on top of the WSE library. On the right, we have the TulaFale script modelling the user-supplied code, together with some library predicates modelling operations performed by WSE. Also on the right, we have the TulaFale tool, which compiles its input scripts into the pure applied pi calculus, which is then analyzed via ProVerif.

TulaFale is a direct implementation of the pi calculus described in a previous formal semantics of web services authentication [BFG04a]. Details of the TulaFale syntax and implementation have been published elsewhere [BFGP04], and the implementation is available for download (`http://securing.ws`). We have successfully used TulaFale to specify and prove (or find errors in) various SOAP security protocols [BFGP04,BCFG04].

2.2 Verifying Policy Configurations

In using TulaFale to verify a specific configuration, we manually model the security checks performed by the client and service deployments in the high-level specification language (the dotted lines in Figure 1). Ideally, however, we would like to directly and automatically verify the configuration files that drive these implementations. Hence, we propose a new language and two new tools to address this problem.

In the absence of an existing XML schema for writing high-level security goals, we design our own simple format for *secure links* between SOAP endpoints hosting sets of principals acting as clients and servers. The link language is

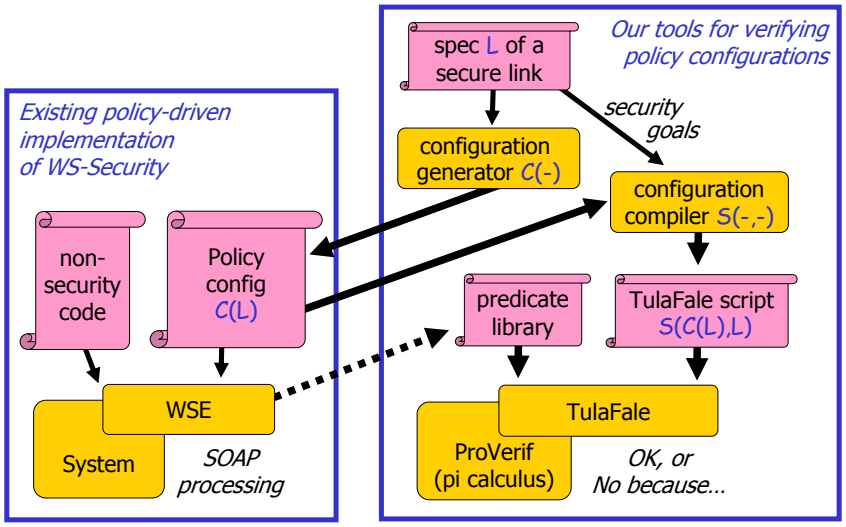

Fig. 2. Generating and Checking Web Services Security Policy Deployments

considerably more abstract (and less expressive) than policies, so that reviewing the security of a link specification is much easier than understanding the security implications of every detail in a configuration.

Both our new tools take a link specification L as input. The first ("configuration generator" in Figure 2) generates WSE policy configurations $\mathcal{C}(L)$ to implement L. The second ("configuration compiler" in Figure 2) generates a TulaFale script $\mathcal{S}(C, L)$, which consists of a formal model of the policy configuration C, plus security goals extracted from L.

For any L, we can check correct generation of $\mathcal{C}(L)$ by compiling to the script $\mathcal{S}(\mathcal{C}(L), L)$, and running the TulaFale verifier. Alternatively, we can use a different (or a modified) configuration C', for instance by handwriting some of the policies, and check that the amended configuration still meets the original security goals, by verifying the script $\mathcal{S}(C', L)$. In this case, we automatically verify formal security guarantees, without the need to manipulate TulaFale scripts. For instance, one could run the verifier whenever the configuration is edited, before committing the changes to a live system.

We have used these tools to analyze and find errors in several policy configurations. Details of the design and implementation of the tools have been published elsewhere [BFG04b].

3 Further Reading

The TulaFale language and implementation is presented through a detailed example in [BFGP04]. The TulaFale implementation is available for download (`http://securing.ws`). The design and implementation of the policy configuration analysis tools are described in [BFG04b].

A longer case study of the TulaFale language and implementation analyzes several scenarios corresponding to the WS-Trust and WS-SecureConversation specifications [BCFG04]. The scripts used in this study are available online.

An earlier work [BFG04a] introduces a preliminary version of TulaFale, defines its semantics via translation into the applied pi calculus [AF01], illustrates TulaFale via several single-message protocols, and describes hand-crafted correctness proofs.

References

AF01. M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In *28th ACM Symposium on Principles of Programming Languages (POPL'01)*, pages 104–115, 2001.

AG99. M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: The spi calculus. *Information and Computation*, 148:1–70, 1999.

BCFG04. K. Bhargavan, R. Corin, C. Fournet, and A. D. Gordon. Secure sessions for web services. In *Proceedings of the 2004 ACM Workshop on Secure Web Services*. ACM Press, October 2004. Full version available at `http://research.microsoft.com/projects/samoa/secure-sessions-with-scripts.pdf`.

BCH+03. D. Box, F. Curbera, M. Hondo, C. Kaler, D. Langworthy, A. Nadalin, N. Nagaratnam, M. Nottingham, C. von Riegen, and J. Shewchuk. Web services policy framework (WS-Policy), May 2003.

BEK+00. D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. Nielsen, S. Thatte, and D. Winer. *Simple Object Access Protocol (SOAP) 1.1*, 2000. W3C Note, at `http://www.w3.org/TR/2000/NOTE-SOAP-20000508/`.

BFG04a. K. Bhargavan, C. Fournet, and A. D. Gordon. A semantics for web services authentication. In *31st ACM Symposium on Principles of Programming Languages (POPL'04)*, pages 198–209, 2004. An extended version appears as Microsoft Research Technical Report MSR–TR–2003–83.

BFG04b. K. Bhargavan, C. Fournet, and A. D. Gordon. Verifying policy-based security for web services. In *Proceedings of the 11th ACM conference on Computer and communications security*, pages 268–277. ACM Press, 2004.

BFGP04. K. Bhargavan, C. Fournet, A. D. Gordon, and R. Pucella. TulaFale: A security tool for web services. In *International Symposium on Formal Methods for Components and Objects (FMCO'03)*, volume 3188 of *LNCS*. Springer, 2004.

BHK+03. D. Box, M. Hondo, C. Kaler, H. Maruyama, A. Nadalin, N. Nagaratnam, P. Patrick, C. von Riegen, and J. Shewchuk. Web services policy assertions language (WS-PolicyAssertions), May 2003.

Bla01. B. Blanchet. An Efficient Cryptographic Protocol Verifier Based on Prolog Rules. In *14th IEEE Computer Security Foundations Workshop (CSFW-14)*, pages 82–96. IEEE Computer Society, 2001.

Bla02. B. Blanchet. From Secrecy to Authenticity in Security Protocols. In *9th International Static Analysis Symposium (SAS'02)*, volume 2477 of *Lecture Notes on Computer Science*, pages 342–359. Springer, 2002.

DHK+04. M. Davis, B. Hartman, C. Kaler, A. Nadalin, and J. Schwarz. *WS–I Security Scenarios*, February 2004. Working Group Draft Version 0.15, at http://www.ws-i.org/Profiles/BasicSecurity/2004-02/SecurityScenarios-0.15-WGD.pdf.

DLHBH+02. G. Della-Libera, P. Hallam-Baker, M. Hondo, T. Janczuk, C. Kaler, H. Maruyama, N. Nagaratnam, A. Nash, R. Philpott, H. Prafullchandra, J. Shewchuk, E. Waingold, and R. Zolfonoon. Web services security policy language (WS-SecurityPolicy), December 2002.

DY83. D. Dolev and A.C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, IT–29(2):198–208, 1983.

GP03. A. D. Gordon and R. Pucella. Validating a web service security abstraction by typing. In *ACM Workshop on XML Security 2002*, pages 18–29, 2003. An extended version appears as Microsoft Research Technical Report MSR–TR–2002–108.

HL03. M. Howard and D. LeBlanc. *Writing secure code*. Microsoft Press, second edition, 2003.

IM02. IBM Corporation and Microsoft Corporation. Security in a web services world: A proposed architecture and roadmap. At http://msdn.microsoft.com/library/en-us/dnwssecur/html/securitywhitepaper.asp, April 2002.

KN+04a. C. Kaler, A. Nadalin, et al. *Web Services Secure Conversation Language (WS-SecureConversation)*, May 2004. Version 1.1. At http://msdn.microsoft.com/ws/2004/04/ws-secure-conversation/.

KN+04b. C. Kaler, A. Nadalin, et al. *Web Services Trust Language (WS-Trust) Version 1.1*, May 2004. At http://msdn.microsoft.com/ws/2004/04/ws-trust/.

Mic02. Microsoft Corporation. *Web Services Enhancements for Microsoft .NET*, December 2002. At http://msdn.microsoft.com/webservices/building/wse/default.aspx.

Mil99. R. Milner. *Communicating and Mobile Systems: the π-Calculus*. Cambridge University Press, 1999.

NKHBM04. A. Nadalin, C. Kaler, P. Hallam-Baker, and R. Monzillo. *OASIS Web Services Security: SOAP Message Security 1.0 (WS-Security 2004)*, March 2004. At http://www.oasis-open.org/committees/download.php/5941/oasis-200401-wss-soap-message-security-1.0.pdf.

NS78. R.M. Needham and M.D. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, 1978.

S+04. J. Schlimmer et al. *A Proposal for UPnP 2.0 Device Architecture*, May 2004. At http://msdn.microsoft.com/library/en-us/dnglobspec/html/devprof.asp.

SS02. J. Scambay and M. Shema. *Hacking Web Applications Exposed*. McGraw-Hill/Osborne, 2002.

Vog03. W. Vogels. Web services are not distributed objects. *IEEE Internet Computing*, 7(6):59–66, 2003.

W3C03. W3C. *SOAP Version 1.2*, 2003. W3C Recommendation, at http://www.w3.org/TR/soap12.