WSEmail: Secure Internet Messaging Based on Web Services

Kevin D. Lux, Michael J. May, Nayan L. Bhattad, and Carl A. Gunter
University of Pennsylvania

Abstract

Web services offer an opportunity to redesign a variety of older systems to exploit the advantages of a flexible, extensible, secure set of standards. In this paper we explore the objective of improving Internet messaging (email) by redesigning it as a family of web services, an approach we call WSEmail. *We illustrate an architecture and describe some applications. Since increased flexibility often mitigates against security and performance, we focus on steps for proving security properties and measuring the performance of our system with its security operations. In particular, we demonstrate an automated proof using TulaFale and ProVerif of a correspondence theorem for an application called on-demand attachments. We also provide performance measures for the basic WSEmail functions in a prototype we have implemented using .NET. Our experiments show a latency of about a quarter of a second per transaction under load.*

1. Introduction

The advent of web services has created the foundation for highly interoperable distributed systems to communicate over the Internet using standardized protocols and security mechanisms. With the foundation now available, older designs and protocols should be reevaluated to see whether they can benefit from the new architecture, standards, and tools. As a case study of such an analysis and redesign, we present *WSEmail*, electronic mail redesigned as a family of web services.

Internet electronic mail (email) is based on a collection of protocols—SMTP, POP, IMAP, S/MIME—that have evolved over a vast installed base. The resulting system has shortcomings in the areas of flexibility, security, and integration with other messaging systems. For instance, problems of authentication and extensibility have plagued attempts to reduce spam, while poor integration with browsers has led to a rash of spoofing attacks ("phishing"). There are many ideas about improving email that can be addressed by new design ideas; a listing of a number of these limitations can be found on the mail-ng list of the Internet Mail Consortium (IMC) (`www.imc.org`). To respond to these shortcomings and pursue new opportunities, we have investigated replacing the existing protocols with protocols based on SOAP, WSDL, XMLDSIG and other XML-based formats. The new protocols are inherently extensible, promise to give stronger guarantees for message authentication, and are amenable to formal modeling and proof.

WSEmail is designed to perform the functions of ordinary email but enable additional security functions and more flexibility. The primary strategy is to import these virtues from the standards and development platforms for web services. Our exploration of WSEmail is based on a prototype architecture and implementation. WSEmail messages are SOAP messages that use web service security features to support integrity, authentication, and access control for both end-to-end and hop-by-hop message transmissions. The WSEmail platform supports the dynamic updating of messaging protocols on both client Mail User Agents (MUAs) and server Mail Transfer Agents (MTAs) to enable custom communications. This flexibility supports the introduction of new security protocols, richer message routing (such as routing based on the semantics of a message), and close integration with diverse forms of communication such as instant messaging.

The benefits of flexibility can be validated by showing diverse applications. However, flexibility often has a high cost for security and performance. This paper focuses on techniques to measure and mitigate these costs for WSEmail. We make two contributions. First, we carry out a case study of a formal analysis of an application called on-demand attachments, in which email with an attachment leaves the attachment on the sender's server rather than placing it on the servers of the recipients. The challenge is to design the associated security for the attachment based on emerging federated identity systems. Second, we carry out a set of experiments intended to determine the efficiency of our base system, including its security operations. Both of these studies demonstrate promise for security and performance for web services in general and WSEmail in particular.

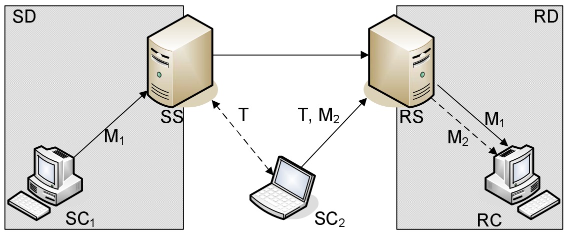

Figure 1. Messaging architecture

The paper is organized in seven sections. Following this introductory section we sketch the architecture of WSEmail focusing on its security assumptions. In the third section we discuss applications we have explored with WSEmail focusing on on-demand attachments. In the fourth section we describe on-demand attachments theoretically and argue that basing email on web services can aid the application of advances in security analysis to new messaging protocols. In the fifth section we discuss our implementation and its performance. The sixth section discusses related work. The seventh section concludes. Interested readers can find more information on our project web page at `wsemail.ws`.

2. Architecture

The baseline protocols for WSEmail are illustrated in Figure 1. In the common case, similar to an SMTP message, an MUA Sender Client SC_1 makes a call on its MTA Sender Server SS to send a message M_1. This and other calls are SOAP calls over TCP; the message M_1 is in the body of the SOAP message and the SOAP header contains information like the type of call and security parameters. The message itself is structured as a collection of XML elements, including, for instance, a subject header. A sample trace of WSEmail messages can be found at `wsemail.ws/messages.html`. After receiving the call from SC_1, the server SS makes a call on the Receiver Server RS to deliver the mail from the Sender Domain SD into the Receiver Domain RD. The Receiver Client RC makes calls to RS to inquire about new messages or download message bodies. In particular, RC makes a call to RS to obtain message headers and then requests the message M_1 if it wishes.

Our design is based on a three-tier authentication system combined with an extensible system of federated identities. The first tier provides user (MUA) authentication based on passwords, public keys, or federated identity tokens. The second tier provides server (MTA) authentication based on public keys with certificates similar to those used for TLS. The third tier uses root certificates similar to the ones in browsers. Overall, this addresses inter-domain authentication in a practical way at the cost of full end-to-end confidentiality. Confidentiality is preserved between hops by TLS or another tunnel protocol. In a basic instance, the message from SC_1 to RC will be given an XMLDSIG signature by SS that is checked by both RS and RC.

The novel aspects of our WSEmail architecture are in the integration and flexibility of the MUA authentication and the ability of both MUAs and MTAs to add new security functions dynamically. To illustrate a variation in the basic protocol, consider our design for instant messaging. Referring again to Figure 1, an instant message M_2 dispatched from a client SC_2 to RC while SC_2 is outside its home domain SD. In this case SC_2 contacts SS to obtain a security token T that will be recognized by RS. Once this is obtained, SC sends M_2 authenticated with this credential to RS and indicates (in a SOAP header) that it should be treated as an instant message by RS and RC. Instant messages are posted directly to the client, with the client now viewed as a server that accepts the instant message call. RS and RC are able to apply access control for this function based on the security token from SC. This token is recognized because of a prior arrangement between SS and RS.

The WSEmail MUA and MTA are based on a plug-in architecture capable of dynamic extensions. Security for such extensions is provided though a policy for trusted sources and the enforcement mechanisms provided by web services. On-demand attachments are an example of such a plug-in, as are a variety of kinds of attachments with special semantics. A party that sends a message with such an attachment automatically includes information for the receiver on where to obtain the software necessary to process the attachment. The client provides hooks for plug-ins to access security tokens, after first performing an access control check on the plug-in. A figure illustrating the MUA (client) components is available at `wsemail.ws/client.html` and screen shots of the GUI can be seen at `wsemail.ws/screenshots.html`. A figure illustrating the server (MTA) components is available at `wsemail.ws/server.html`.

3. Applications

WSEmail offers the possibility to have rich XML formats, extensible semantics on clients and routers, and a range of security tokens. Since there are substantial development platforms for these features from major software vendors, it is easy to use WSEmail as a foundation for a suite of integrated applications that share common code, routing, security, and other features. We have attempted to validate this extensibility by developing several extensions of WSEmail. Our applications include *routed forms* in which a message has a semantic structure that influences its routing and its handling on both MTAs and MUAs, and *integrated instant messaging* in which the same architecture, routing and security are used to provide both email and instant messaging. In this paper we focus on one illustrative application, especially because we wish to show how new protocols can be analyzed for security. This application is known as *on-demand attachments*.

Suppose SC wishes to send an email to a large collection of recipients containing a large document. Suppose, moreover, that only a fairly small, but unknown, subset of the recipients are likely to actually want to look at the document. A straight-forward but expensive approach is to send email to all of the recipients with the document as an attachment. A more efficient approach is to place the document on a web page and send a URL to the recipients. If the document is sensitive, it may be necessary to insist on some access control for the web page, which assumes there is a way to authenticate the recipients if they visit the page to get the document. Systems have been implemented to assist this form of distribution; for instance, it is possible with Microsoft Sharepoint (`www.microsoft.com/sharepoint`) including hooks into the Outlook email client (`office.microsoft.com`).

WSEmail can be used to provide a similar functionality based on federated identities. Figure 2 illustrates such

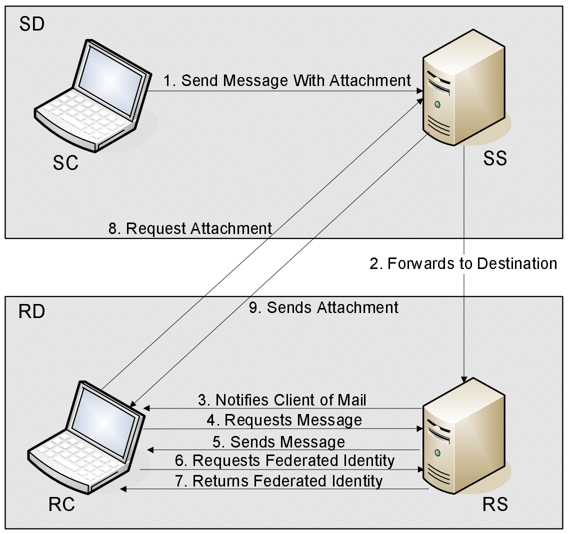
SD
1. Send Message With Attachment
SC
SS
8. Request Attachment
2. Forwards to Destination
RD
9. Sends Attachment
3. Notifies Client of Mail
4. Requests Message
5. Sends Message
6. Requests Federated Identity
7. Returns Federated Identity
RC
RS

Figure 2. On-Demand attachments

a design based on a nine message protocol. In the first message, the client SC sends a WSEmail message with an attachment and an indication that the attachment should remain on the server. For the illustration we assume only one recipient RC, located in another administrative domain RD with its own access control system different from the one in SD, the domain of SC. The server SS in SD accepts the call, stores the attachment, and sends the message and a reference to the attachment to the server RS of the recipient. Clients typically poll their server to inquire about messages. WSEmail can do this or the server can inform the client of messages. In the third, fourth, and fifth messages, RC is informed of the message, requests, and receives it. Upon finding the reference to the attachment at a server in another domain, RC requests a token for communicating with that server and gets one from its server RS. The client RC then uses this token to request and obtain the attachment from SS. We discuss the security analysis of this protocol in the next section.

4. Theory

Trends in the analysis of security protocols will aid the development of WSEmail security by supporting the analysis of web services generally [3] (`securing.ws`) and email specifically [1]. To illustrate this, we describe an on-demand attachments protocol in Figure 1 and state a formally-verified correctness property. We assume familiarity with digital certificates (Γ) and hope the notation is sufficiently self-explanatory. Let S be a public key signature function and H be a one way hash.

Definition: (Public Key Authentication Using a Timed Nonce) We write $A \rightarrow B : M \quad (\text{pkey } \Gamma, r, t)$ to indicate that A sends to B a message of the form: $A \mid M \mid r \mid t \mid S(\text{priv}(\Gamma), H(A \mid M \mid r \mid t))$.

Here t is the current time according to the clock of A and r is a random number selected by A. The principal B processes this message by checking the following conditions in this order: the time t is not older than a given threshold; the validity interval of Γ includes the current time; the nonce r is not in the replay cache of B; the signature checks using public key in Γ; the certificate Γ is trusted. If any of these fails then the remaining steps are omitted and the message is discarded. If all of the conditions succeed, then r is added to the replay cache with an expiration time determined by a given threshold. In this case the message is said to be *valid*. □

Definition: (Salted Password Authentication) We write $A \rightarrow B : M \quad (\text{pswd } P, r, t)$ if A sends B a message of the following form $A \mid M \mid r \mid t \mid \text{MAC}(P, A \mid M \mid r \mid t)$

Here t is the current time according to the clock of A and r is a random number selected by A. The principal B processes this message by checking the following conditions in this order: the time t is not older than a given threshold; the nonce r is not in the replay cache of B; the MAC is correct for the password associated with A. If any of these fails then the remaining steps are omitted and the message is discarded. If all of the conditions succeed, then r is added to the replay cache with an expiration time determined by a given threshold. In this case the message is said to be *valid*. □

Protocol: On-Demand Attachments

Initiation Distribution procedures are used to ensure that the following passwords and keys are known only to the specified principals: principals SC and SS share a password P_{SC}; principal SS has the private key for a certificate Γ_{SS} that is trusted by the other principals; principal RS has the private key for a certificate Γ_{RS} that is trusted by the other principals; principals RS and RC share a password P_{RC}. Principal SC creates a message M that includes the return address SC and an attachment N and sends:

Msg 1 $\mathsf{SC} \rightarrow \mathsf{SS} : \mathsf{SS} \mid (\mathsf{RC} \mid \mathsf{RS}) \mid M \mid N$ (pswd $P_{\mathsf{SC}}, r_1, t_1$)
If SS gets a message of this form, it checks this using the password P_{SC}. If the message is valid, SS selects a locally unique reference N^*, stores the attachment N with this reference and the name SC and sends:

Msg 2 $\mathsf{SS} \rightarrow \mathsf{RS} : \mathsf{RS} \mid (\mathsf{RC} \mid \mathsf{RS}) \mid M \mid (\mathsf{SS} \mid U^*)$ (pkey $\Gamma_{\mathsf{SS}}, r_2, t_2$)
If RS gets a message of this form, it checks this using $\mathrm{priv}(\Gamma_{\mathsf{SS}})$. If the message is valid, RS selects a locally unique reference V^*, stores $M \mid (\mathsf{SS} \mid U^*)$ with this reference and the name SC and sends:

Msg 3 $\mathsf{RS} \rightarrow \mathsf{RC} : \mathsf{RC} \mid (\mathsf{RC} \mid \mathsf{RS}) \mid V^*$ (pkey $\Gamma_{\mathsf{RS}}, r_3, t_3$)
If RC gets a message of this form, it checks this using $\mathrm{priv}(\Gamma_{\mathsf{RS}})$. If the message is valid, RC may send the following request:

Msg 4 $\mathsf{RC} \rightarrow \mathsf{RS} : \mathsf{RS} \mid V^*?$ (pswd $P_{\mathsf{RS}}, r_4, t_4$)
If RS gets a message of this form, it first checks whether V^* is associated with RC. If it is and the message is valid based on the password of RC, then it sends the following message:

Msg 5 $\mathsf{RS} \rightarrow \mathsf{RC} : \mathsf{RC} \mid V^* \mid (\mathsf{SS} \mid U^*)$ (pkey $\Gamma_{\mathsf{RS}}, r_5, t_5$)
If RC receives a message of this form, it checks this using Γ_{RS} and may choose to retrieve the attachment N. The reference V^* is included to ensure that this is the response to Msg 4. To do this it needs a token to authenticate to SS. If it does not have one it may request this by a key pair with K as its public key and sending:

Msg 6 $\mathsf{RC} \rightarrow \mathsf{RS} : \mathsf{RS} \mid \mathrm{token}(K)?$ (pswd $P_{\mathsf{RC}}, r_6, t_6$)
When RS receives a message of this form, it checks validity and creates a certificate Γ_{RC} for use by RC and sends:

Msg 7 $\mathsf{RS} \rightarrow \mathsf{RC} : \mathsf{RC} \mid \Gamma_{\mathsf{RC}}$
When RC receives a message of this form, it checks and stores the certificate for use in accordance with its lifetime. It may choose to obtain the attachment by sending:

Msg 8 $\mathsf{RC} \rightarrow \mathsf{SS} : \mathsf{SS} \mid U^*?$ (pkey $\Gamma_{\mathsf{RC}}, r_7, t_7$)
When SS receives a message of this form, it checks validity using its code for credentials from RS and confirms that the reference U^* is associated with RC. It sends:

Msg 9 $\mathsf{SS} \rightarrow \mathsf{RC} : \mathsf{RC} \mid U^* \mid N$ (pkey $\Gamma_{\mathsf{SS}}, r_8, t_8$)
When RC receives a message of this form, it checks validity and makes the attachment available on RC

We specified this protocol formally using the TulaFale [4] specification language, which has constructs for public key signatures and salted password authentication. The TulaFale script compiles to a script that is verifiable with the ProVerif protocol verifier of Bruno Blanchet (`www.di.ens.fr/~blanchet/crypto-eng.html`, version 1.11) [5]. With this we were able to prove the following correspondence theorem for on-demand attachments: *if* RC *retrieves an on-demand attachment with* SC *as return address, then* SC *sent the attachment.*

The story behind this proof is at least as interesting and important as the theorem itself, given what it illustrates about the potential for formal automated verification of security protocols in general and for secure web services in particular. In our first attempt to do the proof, we formulated the protocol messages and required secrets directly in the ProVerif language. ProVerif's language allows for atoms called events which programmers use to declare that certain actions have happened or conditions have been met. We created events for message sending and receiving as well as the receiving of the attachment. We then queried the correspondence that we desired: RC's event that it received the attachment implies SC's event that it had sent the same attachment. The ProVerif tool ran on our script and quickly output that our correspondence was true.

We realized that there was a problem when we discussed our results with Andrew Gordon and Karthikeyan Bhargavan. Both remarked at the mismatch between the complexity of our protocol (nine messages and four parties) and the speed with which ProVerif returned a result. In their experience ProVerif verifications of complex protocols required time measured in days, not seconds! With some checking and insertion of extra event checks it became clear that we had proven a trivial theorem. RC's "receive attachment" event was unreachable and that was why the prover had declared the correspondence to be true. We had made mistakes in our original formalization and, given our initial "success" with the prover, had not inserted the additional checks necessary to discover if our theorem was trivial.

We decided that rather than attempt to rewrite the protocol in ProVerif's language, we would start from scratch using the TulaFale language and libraries, the aim of which were to help us to focus on just the aspects of the protocol unique to our system. The TulaFale compiler would translate down from the higher level structures to the ProVerif language and then run the ProVerif executable automatically. As expected, this improved confidence and degraded performance. After encoding up to message 5 in the protocol, ProVerif would no longer converge for correspondence checks (that is, receiving Message 5 implies that Message 5 was sent) after two days of run time. The memory footprint for the verifier exploded, taking up to 900MB of memory during execution. Some proofs simply caused an out-of-memory error. We finished the encoding regardless, just ensuring that events were reachable to prevent trivial theorems. When the encoding was complete we attempted to just prove that receiving the attachment implied it was sent, but even after four days of processing on a 2.4GHz computer, we saw no output. With the assistance of Bruno Blanchet, the creator of ProVerif, we found that the proof could be completed with some optimizations—in particular, by discarding the derivation tree ordinarily created by the proof.

Our analysis of on-demand attachments appears to the be the largest application of TulaFale or ProVerif and the largest verification of a any web service security protocol yet completed. It illustrates both the value of these tools and challenges in correct representation and performance. The specification can be found in full at `wsemail.ws/On-Demand.tf` and the output of the proof is at `wsemail.ws/On-Demand.tf.an`.

5. Experiments

Web services are often criticized for being slow based on their design and existing implementation platforms. Security and flexibility also provide a performance challenge. Hence a secure, flexible implementation of messaging based on web services raises concerns about performance. We implemented a prototype for WSEmail as a way to address these concerns at the same time as illustrating the benefits of flexibility. In order to evaluate the efficiency of our messaging system, we built a test bed to stress test our implementation's application and protocols. In this section we describe the implementation, the test bed, and our experiments.

We simulated a real world email environment where many users share a common email server. Users may exchange messages with other users within the local domain or external domains. Users may also interact with their personal inboxes to view and delete messages. For our test we defined four standard email operations: send, list, retrieve, and delete. These operations are discussed in detail below.

5.1. Implementation

Our WSEmail prototype runs on Windows server and client systems. Version 1.0 was implemented over the .NET framework and relies on Web Services Enhancement (WSE) 1.0, CAPICOM 2.00, SQL Server 2000 (to store messages for the server), and IIS 5.0. The current version consists of 68 interfaces and 343 classes organized into 30 projects (see `wsemail.ws/uml.html` for a UML model illustrating the design). About 98% of the software is C# .NET-managed code created with Microsoft Visual Studio. Our instant messaging system also exploits a TLS package from Mentalis (`mentalis.org`) since the .NET platform does not provide native support for TLS. In November 2004 we upgraded WSEmail to version 1.1 in order to get WS-Policy support from Microsoft WSE 2.0. This was challenging because primitive functions from WSE 1.0 that we needed for our WSEmail 1.0 implementation were removed from the WSE 2.0 package forcing us to use both WSE 1.0 and WSE 2.0 to implement WSEmail 1.1.

WSEmail uses DNS SRV records (`ietf.org/rfc/rfc2782.txt`) to determine routing. This makes it possible to run WSEmail over other protocols without changing the way DNS is queried, and we can exploit the priority and weight attributes in the records. These properties of the SRV record allow for future enhancement and present day configuration that is extremely similar to the way SMTP is deployed now.

5.2. Test Bed

Our test bed consisted of a total of four client machines, two mail servers (designated as local and external), one test coordinator and one database/DNS server. The arrangement of the test bed is depicted in Figure 3.

The *test clients* (labeled as T_1 through T_4) all performed operations by sending requests to the "local" email server, S_i. The test client actions were coordinated by the test coordinator, S_{tc}. There also was a second server, S_e, which acted as both an "external" email server and a load generator for the "local" system. S_{db} hosted a message storage database and DNS records for S_i and S_e. The clients all had Pentium 4 2.8GHz processors with 512MB of memory and the Windows XP Pro operating system. They performed four different operations during the test execution: *send* a message to a recipient; *list* the headers of messages in the client's inbox, *retrieve* a particular message, *delete* a particular message. We explored various ways to include a mixture of applications with these basic operations but found it difficult to isolate performance issues clearly in doing this, so we restricted our focus to a demonstration of the basic operations.

The *test coordinator*, S_{tc}, was responsible for distributing the test specifications, starting the test and receiving the results from each client. The coordinator S_{tc} broadcasted its network address, instructing all clients to connect to it and download the test specifications file. The clients then waited for S_{tc} to announce the start of the test, after which the clients executed requests to S_i in compliance with the specs they downloaded. After each client finished, the latencies for each request were reported back to S_{tc}.

The test specifications document described exactly what each client was to do. It indicated whether the client should authenticate using a username token (user name and password) or X.509 certificate. It also specified how many messages were to be sent from each client, to whom are they were to be sent and the size of the message body. The

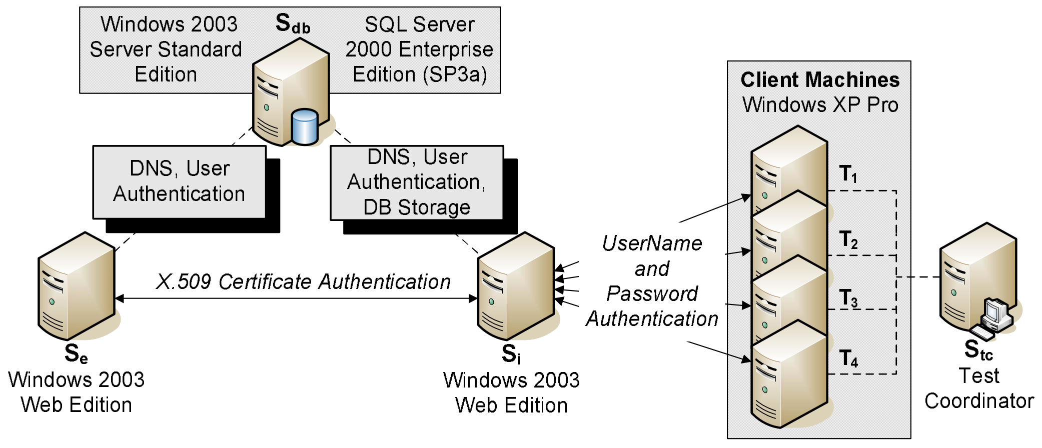

Figure 3. Test bed

specification document also indicated the total number of requests that should be sent and the ratios of the four types of requests.

The *local server* S_i was the focus of our test. It accepted incoming messages from the clients and a *external server* S_e. It performed the necessary authentication, and forwarded external messages to the appropriate destination after performing DNS resolution. If the destination was local (for example, the recipient is on S_i) then the message was stored in S_{db}. If the destination was external, the message was forwarded to S_e. We allowed the local and external server to share a database and DNS server since these were not performance bottlenecks in the system.

The external server S_e played two roles in our test bed. First, it imitated the entire external client list, so that all emails directed to any external client were forwarded to it. On reception of a message addressed to one of the clients that it simulated, it did not save it to the the database server. This was done to prevent S_i from experiencing extra latency due to S_e's database transaction. Rather, it performed the required certificate checking to verify authenticity and then discarded the message. Second, it acted as a load generator and sent one message per second addressed to each of the four clients: T_1 - T_4. These messages were all received by S_i, authenticated, and stored in S_{db}.

5.3. Procedure and Results

The test coordinator S_{tc} provided a test specification document that instructed each client to run one execution thread sending 2,000 requests to S_i. The clients chose send, list, retrieve, and delete operations with 25% chance. In cases where the delete operation was to be performed on an unpopulated inbox, it was considered a no-op and not counted towards the results. However, to avoid this condition, each client's inbox was primed with approximately half a dozen messages. To get the most out of each send event, each message was addressed to both a randomly chosen local client and an external client. The clients were all instructed to authenticate to S_i using username token authentication. S_i and S_e authenticated to each other using X.509 certificate signing. The duration of the entire test was 1826 seconds.

In order to get a client-side view of the efficiency of the system, we measured the latency of each request. A timer was started as the client contacted S_i with a request and stopped after the client received the appropriate response (*e.g.* inbox listing, message received confirmation, and so on). The time difference between the client's request and the server's complete response was the latency of the operation. The results of this calculation point to an average of 0.284 seconds per request with a variance of 0.1389 seconds. The minimum and the maximum latencies were 46.876 ms and 4.0 seconds respectively. Note that the Message Received confirmation does not mean that the message was delivered to the ultimate recipient, just that the message was placed in the delivery queue.

The test results in Table 1 show the throughput of bytes

Table 1. Bytes sent between clients and S_i

Operation	Send	List	Retrieve	Delete
# of requests	1970	2024	2026	1980
% of all requests	24.6	25.3	25.4	24.7
Client to Server Data (MB)	10.74	8.42	8.62	8.4
Server to Client Data (MB)	12.31	324.55	20.41	12.08

sent in MB as a break-down of the number of requests (send, list, retrieve and delete). Therefore the total data in MB from the clients to S_i is 36.18 MB and from the S_i to all the clients is 369.35 MB.

Since each message is also sent to an external client, each send action also sends a message from S_i to S_e. This data is measured according to the representation in Table 2. Therefore the total data exchanged from S_i to S_e is 30.95 MB and from S_e to S_i 30.69 MB.

Table 2. Bytes sent between S_i and S_e

Server name	# of Messages	Send (MB)	Received Confirmations (MB)
S_i	1970	19.59	12.29
S_e	1826	18.40	11.36

The entire test bed data transfer was recorded using the Ethereal network monitor (`www.ethereal.com`), which was run at S_i and S_e. The TCP/IP sessions were then reconstructed using tcpflow (`www.circlemud.org/~jelson/software/tcpflow/`) and post-processed with Perl and awk. Since S_e, acting as a load generator, sent one message per second, 1826 messages were also sent from S_e to S_i over the entire duration of the test. The corresponding byte count represents the messages that were sent and the notification messages that were received.

5.4. Analysis

A best case test of SMTP with no load on server/network and no contention for resources, yielded an average latency of 0.170 ms to send a message of approximately the same size as the WSEmail messages we sent in our experiment. The average difference in latency between WSEmail and the SMTP test is therefore 0.114 ms, which accounts for the additional overhead of the XML parsing and cryptography. In that short time span a large number of operations took place: one secret key signature, one private key signature verification, two public key signatures, and one public key signature verification. Given that the entire system is using XML we conclude that performance is not a barrier to secure web services in this type of application. Indeed, the extra latency probably would not be noticed in a typical client/server environment.

XML and XMLDSIG do have a drawback in their verbosity. Our test bed sent 1 KB mail messages which ballooned into 10 KB responses to the retrieve message action in order to make XMLDSIG work properly. In general, at least 30% of those bytes were the Base64 encoded representations of the certificates used for signing messages. After the certificate size, the WS-Security structures were also a significant amount of overhead, accounting for about 30% of the bytes transferred. WSEmail might need to explore ways to distribute certificates so that they are not replicated excessively. It might also be fruitful to look at how messages are signed to try to minimize their verbosity.

Our experimentation bodes well for web service efficiency, especially for high volume messaging. Extending our experimental results, we find that WSEmail is theoretically capable of handling approximately 1787 messages a minute (combination of incoming and outgoing). We looked for published benchmarks to compare this against and found that the University of Wisconsin-Parkside (`uwp.edu/cgi/netstats.cgi?log=mail`) had a peak usage of 1716 (total of incoming and outgoing) messages per minute over the past year, meaning it should be possible for a single WSEmail server similar to our test system to routinely handle the normal load at that institution. Given the modest cost and capability of our test bed system, we predict that much higher capacities are possible with further hardware investment.

We made some modest adjustments in the WSEmail system to improve its efficiency based on early test results. For instance, our early results show that the client to server bandwidth usage was dominated by list commands. This is because in the test bed the new message-checking mechanism was inefficient. It would download all the headers and compare them to find new ones rather than just asking for messages newer than a certain date. In response to these results, we changed the functionality of WSEmail to ask for new messages by date, increasing its efficiency.

The experimental results shows a promising picture for secure XML-based web services messaging. The responsiveness of the system, combined with the new security features and inherent extensibility, provide a solid framework for new messaging systems. We aim to expand the scope of the test bed to include application tests and use it to regularly confirm performance consequences of changes in the system.

6. Related Work

Work related to WSEmail can be divided into two general areas: improved Internet messaging systems and the analysis of web service security.

Improvements to the SMTP messaging system have often been motivated by two, sometimes overlapping goals: strong message authentication and spam prevention. PGP offers authentication tools that include public/private key signing and encryption. Privacy Enhancement Mail (PEM) (IETF RFCs 1421-4) has mechanisms for privacy, integrity, source authentication, and non-repudiation using public and private key encryption and end-to-end encryption techniques. Zhou, *et. al.* [8] use formal tools to verify the properties of the PEM system. Abadi, *et. al.* [2] use a trusted third party to achieve message and source authentication and formally prove correctness of their protocol.

Changes to the SMTP system aimed at spam reduction include a proposal by Fenton and Thomas [7] which uses public key cryptography and an option for server-signed (rather than client-signed) messages and Petmail (`petmail.lothar.com`). Petmail uses the GPG encryption utility for public key encryption and signing of messages. Users are identified by IDRecords, self-signed binary blobs that include public key, identity, and message routing information. Petmail agents can enforce IDRecord whitelists and policies for contact from first time senders. First-time senders may be forced to obtain tickets from a third party Ticket Server which may perform checks to ensure that the sender is a human (using CAPTCHA reverse Turing tests.) Messages can be encapsulated and sent using SMTP, Jabber, or some other queueing transport protocol. Patterns and options for sender anonymity are offered as well. Our most recent work on extensions of WSEmail show how to do several of these things and more based on WS-Policy negotiations and our dynamic plug-in capability.

In the area of web services security analyses, the Samoa project at Microsoft Research (`securing.ws`) has developed important fundamentals, including a formal semantics for proving web services authentication theorems [3] and the TulaFale language for automating web service security protocol proofs [4]. This is our primary foundation for current and future formal work with WSEmail. Damiani *et. al.* [6] discuss connections between XML, SOAP, and access control languages.

7. Conclusions

We have explored WSEmail, the development of email functions as a family of web services, by developing a prototype system based on an architecture that emphasizes flexibility, security, and integration. We have shown that WSEmail is amenable to the addition of new protocols and the formal analysis of these protocols. We have also shown that the basic WSEmail functions have satisfactory performance. In ongoing work, we are exploring several directions such as: new applications that exploit improved integration between web-like data retrieval functions and the messaging system; challenges to interoperability with a Java implementation of the MUA; and ways to express and negotiate messaging policies. For widespread use, WSEmail faces substantial problems with standardization and interoperability with SMTP, which may be mitigated by writing more plugins like our SMTP-compatible relay agent. However, it is well-suited to some high-security applications even now, offers ideas in exploring the general design space for Internet messaging, and can rely on the standardization advantages of XML as an aid to addressing interoperability challenges. We also aim to support WSEmail on diverse platforms. A project of Heo, Patel, and Shah was partially successful in doing this for a Java WSEmail client based on Sun's JWSDP 1.4 with X.509 security. This went well for some web service standards like WSDL but was challenged by support for some security functions in the message headers such as OASIS UsernameToken authentication.

This work was supported by a gift from Microsoft University Relations, NSF grants CCR02-08996 and EIA00-88028, ONR grant N000014-02-1-0715, and ARO grant DAAD-19-01-1-0473. We are grateful for discussions of WSEmail that we had with Martín Abadi, Raja Afandi, Noam Arzt, Karthikeyan Bhargavan, Luca Cardelli, Dan Fay, Eric Freudenthal, Cedric Fournet, Andy Gordon, Ari Hershl Gordon-Schlosberg, Munawar Hafiz, Jin Heo, Himanshu Khurana, Ralph Johnson, Bjorn Knutsson, Jay Patel, Neelay Shah, Kaijun Tan, and Jianqing Zhang. We are also grateful to Bruno Blanchet for technical support.

References

[1] M. Abadi and B. Blanchet. Computer-assisted verification of a protocol for certified email. In Radhia Cousot, editor, *Static Analysis, 10th International Symposium (SAS'03*, volume 2694 of *Lecture Notes on Computer Scienc*, pages 316–335, San Diego, CA, June 2003. Springer.

[2] M. Abadi, N. Glew, B. Horne, and B. Pinkas. Certified email with a light on-line trusted third party: design and implementation. In *Proceedings of the eleventh international conference on World Wide Web*, pages 387–395. ACM Press, 2002.

[3] K. Bhargavan, C. Fournet, and A. Gordon. A semantics for web services authentication. In *Proceedings of the 31st ACM SIGPLAN-SIGACT symposium on principles of programming languages*, pages 198–209. ACM Press, 2004.

[4] 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)*, LNCS. Springer, 2004.

[5] B. Blanchet. An efficient cryptographic protocol verifier based on prolog rules. In *Proceedings of the 14th IEEE Workshop on Computer Security Foundations*, page 82. IEEE Computer Society, 2001.

[6] E. Damiani, S. De Capitani di Vimercati, and P. Samarati. Towards securing XML Web services. In *Proceedings of the 2002 ACM workshop on XML security*. ACM Press, 2002.

[7] J. Fenton and M. Thomas. Identified internet mail. Work in Progress draft-fenton-identified-mail-01, IETF Internet Draft, October 2004. Expires April 2005.

[8] D. Zhou, J. Kuo, S. Older, and S. Chin. Formal development of secure email. In *Proceedings of the 32nd Hawaii International Conference on System Sciences*. IEEE Computer Society, 1999.