Chapter 14

FORENSIC WEB SERVICES

Murat Gunestas, Duminda Wijesekera and Anoop Singhal

Abstract Choreography, orchestration and dynamic invocation allow new web services to be composed from existing ones. However, these compositions create service interdependencies that can be misused for financial fraud and other illegal purposes. When a misuse is reported, investigators have to navigate through collections of logs to recreate the invocation scenario in order to evaluate the misuse claims. We propose the creation of forensic web services that can securely maintain transaction records between web services. An independent entity could use the stored records to reproduce the complete transaction history when investigating a misuse claim.

Keywords: Web services, service oriented architecture, transaction forensics

1. Introduction

Web services are being used for many commercial, government and military purposes. New web services can be created by seamlessly integrating existing web services using techniques such as choreography, orchestration, dynamic invocation and brokering. These service-level compositional techniques create complex dependencies between web services of different organizations. When they are exploited, multiple servers and organizations are affected, resulting in considerable financial loss and infrastructure damage.

Investigating web service incidents requires that the dependencies between service invocations be retained in a neutral and secure manner so that the alleged activity can be recreated while preserving evidence that could lead to and support prosecution. Evidence extracted from web servers, such as XML firewall alerts from endpoint services and web server log records, have limited forensic value. Defendants can claim

that they did not send the messages in question or that the plaintiffs altered the logs to make their cases.

To address these issues, we propose Forensic Web Services (FWSs), which preserve the evidence needed to recreate composed web service invocations independent of the parties with vested interests. Note that a simple non-repudiation argument with multiple log records has no forensic value. Also, web service forensics cannot be treated as a bilateral problem between two web services because of dynamic compositions. Consequently, FWSs provide web-service-based forensic capabilities to web services. This requires FWSs to be integrated with web services that require them; these web services are referred to as customer web services. In order to do so, FWSs provide a centralized service access point to customer services. The information retained by FWSs as a trusted third party can be provided to forensic examiners. Previous proposals for monitoring web services [6] and generating evidence [10, 16, 26] are primarily for business purposes. We are not aware of similar applications that support forensic investigations.

Organizations that are tightly integrated through web transactions and processes can benefit from FWSs in many ways. They can hold their partner services accountable when their vulnerabilities affect transaction confidentiality, availability, etc. Also, the details of malicious activity can impact punitive measures and damage claims. We show that non-repudiable logging of critical information exchanges is an effective way to meet these needs. Some logging and processing approaches already exist for web services [5, 6, 28]. Also, Sremack [30] has proposed an approach for conducting online investigations. However, these approaches do not employ a trusted third party to generate and preserve evidence or offer conclusive evidence as provided by the FWS framework.

2. Web Service Attacks

Numerous web service attacks, such as WSDL/UDDI scanning, parameter tampering, replays, XML rewriting, man-in-the-middle attacks, eavesdropping and routing detours have been identified [7–9, 17, 18, 20, 27] and characterized (see, e.g., [31, 33]). To motivate the need for FWSs, we discuss the cross-site scripting (CSS) attack [9], which is presented in Figure 1. An attacker with stolen credentials injects malicious data, invoking an update operation on a weather service that stores a script (including instructions to steal cookies) in web browsers. Next, a web application, say Portal Web Application in Figure 1, retrieves this malicious data when invoking a get operation and publishes weather information to its subscribers in HTML, thereby making the

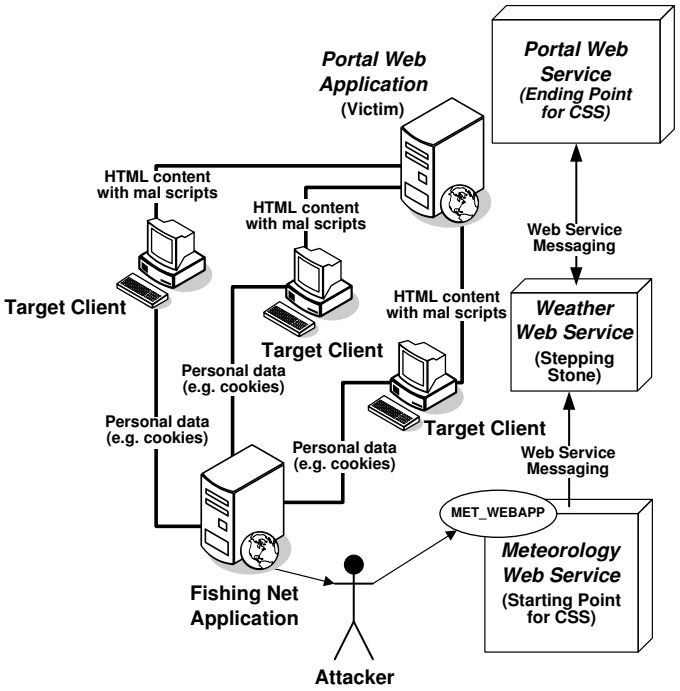

Figure 1. Cross-site scripting attack.

subscribers send their personal information (stored in cookies) to the attacker's Fishing Net Application.

3. Forensic Web Services Framework

The Forensic Web Services (FWSs) framework consists of two services. One generates pairwise evidence when transactions occur between pairs of web services. The other composes evidence generated from pairwise transactions and creates complex transaction scenarios on demand.

The FWSs framework uses trusted third parties that sit in between transactions. To obtain forensic services, all web services must sign up with a FWS (Figure 2), and all FWSs agents must cooperate by providing relevant pairwise transactional evidence that they have stored. Four roles are involved in the process: sender, receiver, operator FWS and non-operator FWS. The operator FWS refers to a FWS selected by either party to manage the steps listed below; the non-operator FWS belongs to the other party. A FWSs registry is available to locate all registered FWSs servers. FWSs systems must satisfy the following requirements:

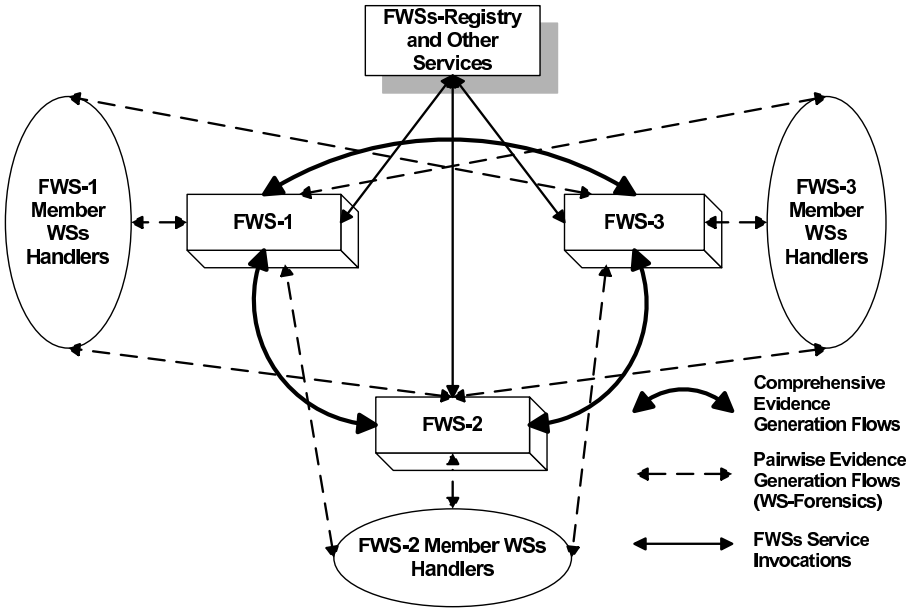

Figure 2. Forensic Web Services framework.

- The web service call stack must be enriched with a WS-Forensics layer.
- A message format is required to communicate with WS-Forensics layer messages and store them in FWSs servers.
- All web services must use a client agent that re-routes their transactional messages through FWSs servers.
- The underlying system must provide a trust base and cryptographic services.

The web service stack has three layers: the bottom layer consists of SOAP messages, the middle layer WS-SecureConversations and the top layer WSDL specifications (Figure 3). A forensic layer is added between the middle and top layers to re-route transactions through FWSs servers. The sender web service and receiver web service communicate using their WSDLs independently of the underlying WS-Forensics layer.

WS-Forensics uses the following message format:

$$\langle \#\text{session}|\#\text{message}|\#\text{signatureK}(\#\text{session}|\#\text{message/sequence}|\#\text{message/envelope})\rangle$$

where # refers to the points in XML format, | denotes concatenation and / points to the subparts of elements. Also, "session" identifies a

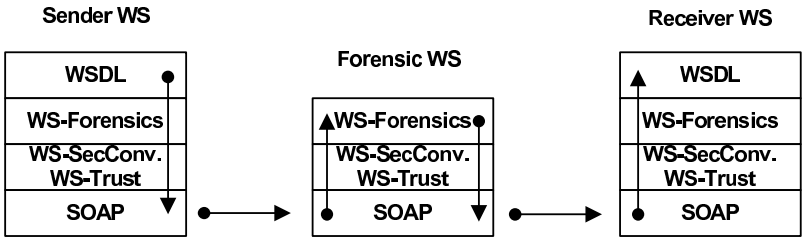

Figure 3. WS-Forensics stack.

WS-Forensics conversation and "message" is the upper layer content and its sequence number in the conversation. Both endpoints (sender and receiver) sign a session.

FWSs store messages in two formats: LogRecordIndex (LRI) and LogRecord (LR) without signatures, where LRI records a single fwsMessage, LR stores entire WS-Forensics sessions, including all fwsMessages delivered to and/or generated by FWSs. LRIs are stored at both endpoints while LRs are stored only at the operator FWS. FWSs also timestamp all messages. LR contains a record index with the final timestamp, status and the last sequence value of the conversation. All transaction information is reliably intercepted and re-routed through FWSs servers using sender and receiver processes positioned in front of each web service endpoint. The sender process uses the FWSs handler exposed by the forensics layer, adds the extra routing information to conform with the WS-Forensics message format and passes it to the WS-SecureConversation or WS-Trust handler exposed by the WS-Trust layer. Similarly, the receiver process verifies the signatures, extracts the SOAP message and passes it to the intended service or port type.

WS-Forensics is designed to run over a secure layer with the following services (that are already satisfied by WS-Trust [21] and WS-SecureConversation [22]):

- **Authentication:** Senders, receivers and FWSs nodes.
- **Delegated Authentication:** As a trusted third party, FWSs nodes authenticate themselves to the receiver on behalf of the sender.
- **Channel Confidentiality and Integrity:** FWSs nodes must ensure the confidentiality and integrity of channels between senders and receivers.
- **Reliability:** Messages in channels between FWSs nodes and customer nodes must be reliable.

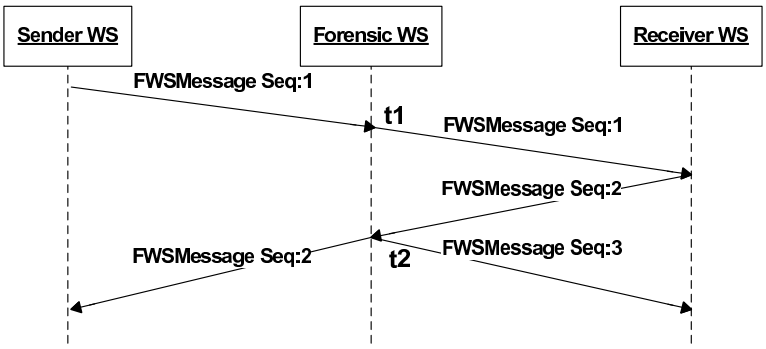

Figure 4. Operator FWS managing SELP.

4. Collecting Pairwise Evidence

FWSs collect pairwise transactional evidence using the Simple Evidence Layer Protocol (SELP) [10]. Four roles are employed: sender, receiver, operator FWS and non-operator FWS. The following steps are performed by the operator FWS (see Figures 4 and 5):

1. The operator FWS receives MsgSeq.1, which contains ⟨#session|#message|#signatureSender-K(#session|"1"|#env)⟩.

2. It validates and stores the message, creates an LR and LRI for MsgSeq.1 and notifies the non-operator FWS.

3. It forwards MsgSeq.1 to the receiver and starts a timer.

4. If the response MsgSeq.2 does not reach in time, the operator FWS signs MsgSeq.-1(⟨#session|#message|#signatureFWS-K(#session|"-1"|#env)⟩), stores the message and sends it back to the sender; also, it creates an LRI, which is sent to the non-operator FWS. However, if MsgSeq.2 (⟨#session|#message|#signatureReceiver-K(#session|"2"|#env)⟩) arrives on time then, it stores the message and forwards it to the sender; also, it creates an LRI, which is sent to the non-operator FWS.

5. It creates, signs and sends MsgSeq.3 (⟨#session|#message|#signatureFWS-K(#session|"3"|#env)⟩) to the receiver. It also stores the message in the LR and sends the LRI to the non-operator FWS. Dependencies between stored data are maintained using LRIs.

The SELP protocol and FWSs event logs retain the evidence required to verify the sender's claims of timely transmission, the receiver's claims

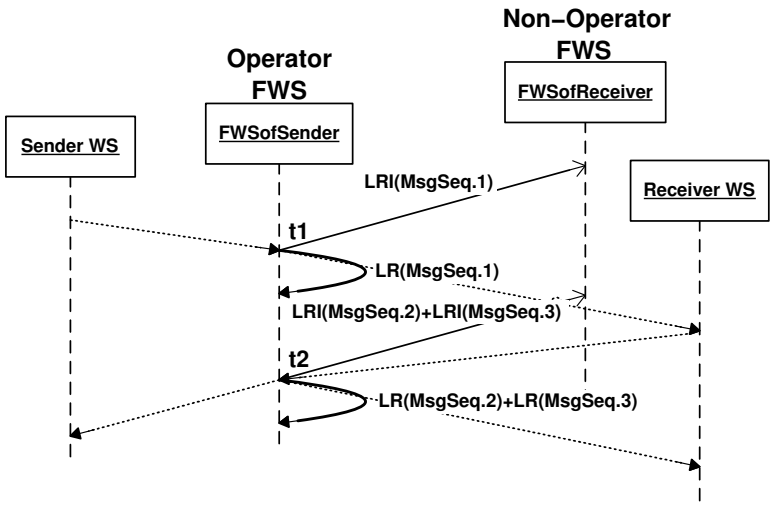

Figure 5. Operator FWS storing pairs.

not to have received messages in a timely manner, either party's claims of non-availability of the other party, and any contractual violations.

5. Creating Evidence for Scenarios

This section describes the main data types and algorithms used to collect and preserve evidence of pairwise transactions involving web services.

FWSs store information exchanged between pairs of services in LRI tables that are used to generate service dependencies expressed as a dependency graph. Dependency graph nodes have the complex type *WebServiceNode*, where each *WebServiceNode* has a unique ID and field *NodeLevel*, which expresses the degree of adjacency of the node to the root of the graph. The field *NodeThreshold* expresses the maximum degree of adjacency from the root. The edges of the graph are represented using the complex data type *LogRecordEdge* with *SenderID* and *ReceiverID* attributes. An authorized requestor generates evidence bags by providing the required arguments using *generateEvidenceBagPortType*, a port that calls other FWSs to collect dependent evidence residing in its log records. FWSs use *StorageService*, *MembershipQueryService*, *SecurityService*, *EvidenceBagService* and other auxiliary internal services. The FWSs Registry manages the member registration processes. The operation *getFWSPortType* called by the FWSs nodes retrieves the ID

```
1.  partnerLinks: SecurityService; VirusScannerService;
        SignatureDetectionSrv; RootFWS; Requestor; FWSRegistry
2.  variables: EvidenceBagIn; EvidenceBagOut; LogRecordEdges;
        DependentsBagIn; DependentsBag;
        LogRecordEdgesForEvidenceGraph;
3.  begin
4.      receive EvidenceBagIn from Requestor
5.      invoke getFWSs(RootWS) in FWSRegistry
6.      assign RootFWS partnerLink
7.      assign EvidenceBagIn to DependentsBagIn
8.      invoke collectDependents(DependentsBagIn) in RootFWS
9.      assign DependentsBag to LogRecordEdges
10.     assign distinct ArrayOfFWSTTP from LogRecordEdges
        ←!– Invokes a set of FWSTTPs to get actual LREs by their LRIs →
        ←!– using flowN loop structure →
11.     flowN N='countNodes(' ArrayOfFWSTTP '...)' indexVariable='index'
12.         partnerLink: OwnerFWSOfLogRecords
13.         variables: LogRecordEdgesOutput
14.         assign OwnerFWSOfLogRecords partnerLink
15.         invoke getLogRecordsByValue in OwnerFWSOfLogRecords
16.         receive LogRecordEdgesOutput as getLogRecordsByValue callback
            ←!———————— Stores the result ————————→
17.         append LogRecordEdgesForEvidenceGraph
                from LogRecordEdgesOutput
18.     end of flowN
19.     assign LogRecordEdgesForEvidenceGraph to EvidenceBagOut
20.     invoke scan(EvidenceBagOut) in VirusScannerService
21.     invoke detect(EvidenceBagOut) in SignatureDetectionSrv
22.     invoke signAndEncrypt(EvidenceBagOut) in SecurityService
23.     reply EvidenceBagOut to Requestor
24. end
```

Figure 6. Building evidence bags.

of the registered FWS of any web service. The operation *registryInfoPortType* is used for member registration and de-registration.

FWS build digital evidence bags [11] using the pseudo BPEL algorithm presented in Figure 6 [14]. First, a requestor starts building digital evidence bags (Line 4) by invoking the *generateEvidenceBag* process with the suspected root *WebServiceNode*, *StartTime* (defines the start time), *TimeThreshold* (defines the time range) and *NodeThreshold* (defines the node range), which are included in the *EvidenceBagIn* message. In Lines 5–8, the FWS gets the FWS that controls the root node, which is the start point for the collection process. The FWS assigns the address of the rootFWS partner link and continues by invoking the *collectDependents* process, which runs recursively over many FWSs with the *DependentsBagIn* message as parameter (see Figure 7 for details). *Depen-*

```
    ←!– Starts extracting values (timeThreshold, nodeThreshold, etc.) →
    ←!– from DependentsBagIn and initializes creating the →
    ←!– WebServiceNodes and LogRecordEdges instances of GRAPH →
1.  baseTime = startTime - timeThreshold
2.  for each logRecordIndex LRI in FWS {
3.      timeThreshold=timeTreshold - (startTime - LRI.timeStamp)
4.      startTime=LRI.timestamp
5.      for each webServiceNode WS in GRAPH {
6.          if (SenderWS | ReceiverWS ∈ LRI & LRI ∉ GRAPH
                & R.timestamp ≥ baseTime & WS.nodeLevel
                ≤ WS.nodeThreshold) {
7.              Add LRI as edge into GRAPH
8.              if (LRI's partner web service PWS ∉ GRAPH) {
9.                  PWS.nodeLevel=WS.nodeLevel+1
10.                 PWS.nodeThreshold=nodeThreshold
11.                 Add the PWS into GRAPH }
12.             if (LRI's PWS ∉ this.FWS & LRI's PWS ∉ GRAPH) {
13.                 NeighbourFWS = getFWS(PWS)
14.                 NeighbourFWS.collectRecords(DependentsBagIn)
15.                 Merge DependentsBagOut into GRAPH}}}}
16. return GRAPH in DependentsBagOut format
```

Figure 7. Collecting dependent processes.

dentsBagIn is assigned the values in the *EvidenceBagIn* message in Line 7. The FWS is returned a final *DependentsBag* message by the children of the recursive call and the returned message contains *LogRecordEdges* only with index information (LRI). In order to convert *LogRecordEdges* with LRIs into *LogRecordEdges* containing LRs (actual contents of messages), the *generateEvidenceBag* process first extracts distinct *fwsttp*s from *LogRecordEdges* into an array in Line 10. In Lines 11–8, the flowN structure in BPEL is used to create dynamic parallel execution scopes for each *fwsttp.location*. For each *fwsttp.location*, dynamic partner links *OwnerFWSOfLogRecords* are also created. Then, *getLogRecordsByValue* operations in these partner links are invoked for each parallel scope and the results are combined in the *LogRecordEdgesForEvidenceGraph*. *LogRecordEdgesForEvidenceGraph* is assigned to *EvidenceBagOut*, which constitutes the actual *EvidenceGraph* document. Other bookkeeping procedures such as scanning and signature verification are applied between Lines 20 and 22. Finally, a response is sent to the requestor in Line 23.

The *generateEvidenceBag* process in Figure 6 is a wrapper of the *collectDependents* process inspired by King and Chen's dependency graph algorithm [15] and Wang and Daniels' evidence graph generation algorithm [32]. The algorithm in Figure 6 first creates instances of *WebSer-*

viceNodes and *LogRecordEdges* and loads the *DependentsBagIn* message into these objects setting the *WebServiceNode* part as a root node for the execution of the algorithm. All the other values in the input message are loaded into the corresponding variables (e.g., *timeThreshold* and *nodeThreshold*). After the initialization phase, the algorithm listed in Figure 7 is used. The created objects *WebServiceNodes* and *LogRecordEdges* are the nodes and edges of the dependency graph (GRAPH). The GRAPH is constructed based on two facts. First, the algorithm traverses the LRIs in decreasing order of time to search for dependent web service nodes among the sender/receiver fields of the log records; these are inserted into *LogRecordEdges*, which sets the *SenderID*, *ReceiverID* and *DependencyDirection* attributes provided their timestamp is within the time threshold. Second, when a new partner web service is found in the LRIs, it is added to the *WebServiceNodes* object only if the current web service node's *nodeLevel* is less or equal to *nodeThreshold*.

6. Related Work

To the best of our knowledge, no distributed forensic framework exists for investigating interrelated web services. However, the research efforts discussed in this section share some common features with our objectives and/or methods.

The FWSs design is influenced by WS-NRExchange [26], especially its implementation of fair non-repudiation using the Coeffey-Saidha protocol [4]. However, unlike FWSs, WS-NRExchange does not address choreography and service compositions.

Herzberg and Yoffe [10] proposed the use of an evidence layer for e-commerce transactions located on top of the transport layer. The FWSs framework incorporates their SELP specification, which was designed for the evidence layer.

FWSs use trusted third parties for pairwise evidence generation as do Coffey and Saidha [4]. Certified e-mail protocols [19] have also been used without trusted third parties [16]. Onieva and co-workers [23] proposed the use of inline trusted third parties for e-commerce transactions with multi-recipient cases through these intermediaries, but not for forensic applications. Bilal and colleagues [3] have used BPEL to implement a non-repudiation protocol for web services; however, their solution does not use trusted third parties and, therefore, lacks message content handling capabilities.

FWSs use handlers in an existing web service architecture [1, 2, 12, 24]. Axis2 [25] also implements web service standards such as Rampart

for WS-SecureConversation, Rahas for WS-Trust, Sandesha2 for WS-RM, and Kandula for WS-Coordination [13].

WSLogA [5] tracks web service invocations by logging them using SOAP intermediaries. However, unlike FWSs, it does not have a distributed collection mechanism for gathering comprehensive forensic evidence from services sharing multiple servers.

Research in the area of network forensics has also inspired the design of the FWSs framework. These include Wang and Daniels' use of intrusion detection system alerts to generate evidence graphs for network forensic analysis [32] and ForNet's use of router logs in its distributed network forensics framework [29].

7. Conclusions

Composed, choreographed and stand-alone web services span many applications. Consequently, the exploitation of a vulnerability in one service can impact many other services. The Forensic Web Services framework supports the investigation of attacks and the assignment of blame. This capability is provided as a service to other web services by logging service invocations. All the logged data is preserved in a digital evidence bag, which can be used to recreate attacks in a forensically-sound manner.

References

[1] Apache Software Foundation, Axis2 Architecture Guide (ws.apache .org/axis2/0_95/Axis2ArchitectureGuide.html), 2006.

[2] BEA Systems, Specifying SOAP handlers for a web service, BEA WebLogic Workshop Help (Online), San Jose, California (edocs.bea .com/workshop/docs81/doc/en/core/index.html).

[3] M. Bilal, J. Thomas, M. Thomas and S. Abraham, Fair BPEL processes transaction using non-repudiation protocols, *Proceedings of the IEEE International Conference on Services Computing*, pp. 337–340, 2005.

[4] T. Coffey and P. Saidha, Non-repudiation with mandatory proof of receipt, *ACM SIGCOMM Computer Communication Review*, vol. 26(1), pp. 6–17, 1996.

[5] S. da Cruz, L. Campos, M. Campos and P. Pires, A data mart approach for monitoring web services usage and evaluating quality of services, *Proceedings of the Twenty-Eighth Brazilian Symposium on Databases*, 2003.

[6] S. da Cruz, M. Campos, P. Pires and L. Campos, Monitoring e-business web service usage through a log based architecture, *Proceedings of the IEEE International Conference on Web Services*, pp. 61–69, 2004.

[7] Y. Demchenko, L. Gommans, C. de Laat and B. Oudenaarde, Web services and grid security vulnerabilities and threats analysis and model, *Proceedings of the Sixth IEEE/ACM International Workshop on Grid Computing*, 2005.

[8] S. Faust, SOAP web services attacks (www.net-security.org/dl/articles/SOAP_Web_Security.pdf), 2003.

[9] D. Green, Attacking and defending web services, presented at the *Nebraska CERT Conference* (www.certconf.org/presentations/2006/files/TA2.pdf), 2006.

[10] A. Herzberg and I. Yoffe, The Delivery and Evidence Layer, Report 2007/139, Cryptology ePrint Archive (eprint.iacr.org/2007/139.pdf), 2007.

[11] C. Hosmer, Digital evidence bag, *Communications of the ACM*, vol. 49(2), pp. 69–70, 2006.

[12] IBM Corporation, JAX-RPC handlers collection, Armonk, New York (publib.boulder.ibm.com/infocenter/wasinfo/v6r0/index.jsp?topic=/com.ibm.websphere.pmc.express.doc/sibusresources/JAXRPC Handler_CollectionForm.html), 2007.

[13] C. Jayalath and R. Fernando, A modular architecture for secure and reliable distributed communication, *Proceedings of the Second International Conference on Availability, Reliability and Security*, pp. 621–628, 2007.

[14] M. Juric, *Business Process Execution Language for Web Services*, Packt Publishing, Birmingham, United Kingdom, 2006.

[15] S. King and P. Chen, Backtracking intrusions, *ACM SIGOPS Operating Systems Review*, vol. 37(5), pp. 223–236, 2003.

[16] S. Kremer, O. Markowitch and J. Zhou, An intensive survey of fair non-repudiation protocols, *Computer Communications*, vol. 25(17), pp. 1606–1621, 2002.

[17] J. Mallery, J. Zahn, P. Kelly, W. Noonan, E. Seagren, P. Love, R. Kraft and M. O'Neill, *Hardening Network Security*, McGraw-Hill/Osborne, Emeryville, California, 2005.

[18] M. McIntosh and P. Austel, XML signature element wrapping attacks and countermeasures, *Proceedings of the Second ACM Workshop on Secure Web Services*, pp. 20–27, 2005.

[19] S. Micali, Certified e-mail with invisible post offices, presented at the *Sixth Annual RSA Data Security Conference*, 1997.

[20] W. Negm, Anatomy of a web services attack: A guide to threats and preventative countermeasures (www.bitpipe.com/detail/RES/1084293354_294.html), 2004.

[21] OASIS Web Services Secure Exchange Technical Committee, WS-Trust V1.0, OASIS (www.oasis-open.org/committees/download.php/16138/oasis-wssx-ws-trust-1.0.pdf), 2006.

[22] OASIS Web Services Secure Exchange Technical Committee, WS-SecureConversation 1.3, OASIS (docs.oasis-open.org/ws-sx/ws-secureconversation/200512/ws-secureconversation-1.3-os.html), 2007.

[23] J. Onieva, J. Zhou, M. Carbonell and J. Lopez, Intermediary non-repudiation protocols, *Proceedings of the IEEE International Conference on E-Commerce Technology*, pp. 207–214, 2003.

[24] Oracle, Using JAX-RPC handlers, *Oracle Application Server Web Services Developer's Guide*, Redwood Shores, California (download.oracle.com/docs/cd/B31017_01/web.1013/b28974/jaxrpchandlers.htm), 2006.

[25] S. Perera, C. Herath, J. Ekanayake, A. Ranabahu. D. Jayasinghe, S. Weerawarana and G. Daniels, Axis2: Middleware for next generation web services, *Proceedings of the IEEE International Conference on Web Services*, pp. 833–840, 2006.

[26] P. Robinson, N. Cook and S. Shrivastava, Implementing fair non-repudiable interactions with web services, *Proceedings of the Ninth IEEE International EDOC Enterprise Computing Conference*, pp. 195–206, 2005.

[27] J. Rosenberg and D. Remy, *Securing Web Services with WS-Security: Demystifying WS-Security, WS-Policy, SAML, XML Signature and XML Encryption*, Sams Publishing, Indianapolis, Indiana, 2004.

[28] M. Rouached and C. Godart, Analysis of composite web services using logging facilities, *Proceedings of the Second International Workshop on Engineering Service-Oriented Applications: Design and Composition*, pp. 74–85, 2006.

[29] K. Shanmugasundaram, N. Memon, A. Savant and H. Bronnimann, ForNet: A distributed forensics network, *Proceedings of the Second International Workshop on Mathematical Methods, Models and Architectures for Computer Networks Security*, pp. 1–16, 2003.

[30] J. Sremack, Investigating real-time system forensics, *Proceedings of the First International Conference on Security and Privacy for Emerging Areas in Communication Networks*, pp. 25–32, 2005.

[31] A. Vorobiev and J. Han, Security attack ontology for web services, *Proceedings of the Second International Conference on Semantics, Knowledge and Grid*, p. 42, 2006.

[32] W. Wang and T. Daniels, Building evidence graphs for network forensics analysis, *Proceedings of the Twenty-First Annual Computer Security Applications Conference*, pp. 254–266, 2005.

[33] W. Yu, P. Supthaweesuk and D. Aravind, Trustworthy web services based on testing, *Proceedings of the IEEE International Workshop on Service-Oriented System Engineering*, pp. 159–169, 2005.