Journal of Computer Science

Original Research Paper

# SSOAM: Automated Security Testing Framework for SOA Middleware in Banking Domain

**Mustafa Al-Fayoumi, Ruba Haj Hamad and Jaafer Al-Saraireh**

*Department of Computer Science, Princess Sumaya University for Technology, Amman, Jordan*

*Article history*
Received: 0402-2018
Revised: 17-05-2018
Accepted: 11-07-2018

Corresponding Author:
Mustafa Al-Fayoumi
Department of Computer Science, Princess Sumaya University for Technology, Amman, Jordan
Email: m.alfayoumi@psut.edu.jo

**Abstract:** In the banking domain, a high level of security must be considered and achieved to prevent a core-banking system from vulnerabilities and attackers. This is especially true when implementing Service Oriented Architecture Middleware (SOAM), which enables all banking e-services to be connected in a unified way and then allows banking e-services to transmit and share information using simple Object Access Protocol (SOAP). The main challenge in this research is that SOAP is designed without security in mind and there are no security testing tools that guarantee a secure SOAM solution in all its layers. Thus, this paper studies and analyzes the importance of implementing secure banking SOAM design architecture and of having an automated security testing framework. Therefore, Secure SOAM (SSOAM) is proposed, which works in parallel with the banking production environment. SSOAM contains a group of integrated security plugins that are responsible for scanning, finding, analyzing and fixing vulnerabilities and also forecasting new vulnerabilities and attacks in all banking SOAM layers.

**Keywords:** SOA Middleware, BPEL, Automation Security Testing Framework, Orchestrated Business Process, SOAP Protocol, Secure Banking Architecture

## Introduction

Service-Oriented Architecture Middleware (SOAM) is considered a paradigm shift in designing banking infrastructure that allows core banking to integrate heterogeneous and legacy systems or e-services to innovate a bank's service delivery and guarantee efficient business process management and orchestration.

In the banking domain, the real competition is to stay innovative, competitive and cost-effective and to fulfill customers' needs, in order to achieve greater profitability. SOAM is an open, extensible and compassable architecture that provides services interoperability and reusability. Furthermore, SOAM allows enterprises to apply the agile methodology and become flexible during their application development cycle (Hariharan and Babu, 2014).

Banking solutions are rarely implemented in a standalone way and must have the ability to securely, inter-operate and integrate different data sources and their systems such as mobile banking, internet banking, electronic check clearing (ECC), ATMs and others. Each of these e-services will have been built by various vendors using different programming languages such as .NET and JAVA, so managing and controlling these e-services is a serious challenge for any bank, especially when the bank wants to achieve service quality, high performance, availability, customer's satisfaction and a high level of security. Thus, having an integration layer is a must in the banking domain: This is SOAM. The SOAM integration layer allows banks to deploy new services quickly and with lower cost (Early and Free, 2002; Keen *et al*., 2017).

The study in this study is based on a real example of an SOAM banking project. A similar approach may be taken in another integration projects to an extent depending on the e-services and the requirements of the banking system.

The rest of the paper is organized as follows. Section 2 discusses the literature review and their considerations. An overview of service oriented architecture is presented in section 3. Section 4 discusses the motivation of the current study. The proposed SSOAM architecture design and its related work mechanism is presented in section 5 and section 6 discusses SSOAM security analysis and evaluations then finally section 7 contains the conclusion and suggestions for future work.

## Literature Review

With regard to SOAM security there are many suggested plugins, patterns, framework and automated tools for achieving a high level of security. The most popular was proposed by Erl (2009). This considers data confidentiality and data origin authentication, which is used for message security level; it also considers direct and brokered authentication that uses an access control concept to allow specific users to access services with special permissions. However, the author did not study security issues related to the communication channel or data encryption pattern and did not consider the idea of building a secure banking architecture by using SOAM (Erl, 2009).

### *WS Attacker*

Web Service Attacker was proposed by Morris *et al.* (2010). This tool is used for penetration tests for XML based web services. It mainly checks any new vulnerability in the SOAM and supports the building of a secure service. This plugin focuses on the individual business process by analyzing the Web Service Description Language (WSDL) file. The main weakness in this plugin, which the author did not consider is that the orchestrated services, located in business process execution language (BPEL) component, could not prevent XML interpreting tags and replay attacks (Hariharan and Babu, 2014; Mainka *et al*., 2012).

### *ATLIST*

Attentive Listener was proposed by Lowis and Accorsi (2011). This plugin performs vulnerability analysis on orchestrated business processes within SOAM, but was tested in a test lab not in parallel with a banking production environment for 24 h per 7 days a week. The authors did not run this plugin on the production environment and did not consider any performance issues while transmitting data (Lowis and Accorsi, 2011).

### *TulaFale*

This plugin was proposed by Bhargavan *et al.* (2004). It is effective for standards XML files such as WSDL files, but the authors did not consider how to handle some headers security functions such as OASIS UsernameToken authentication (Bhargavan *et al*., 2004; Bhargavan and Gordon, 2017; Bhargavan *et al*., 2007; Lux *et al*., 2005).

### *SOFIA Oracle*

A new security plugin was proposed by Ceccato *et al.* (2016). This is related mainly to the communication channel with the database to protect any system from SQL injection attack (SQLi). This plugin needs to improve its performance to speed up its work cycle.

In addition, a group of research papers studied different testing types for SOAM technology such as unit, functional and regression testing but did not directly discuss automated security testing, did not focus on the importance of scanning the production environment to report vulnerabilities and attackers dynamically and did not connect the complete solution with a business intelligence tool to forecast and predict new vulnerabilities and attackers (Kajtazovic *et al*., 2017; Inaganti and Aravamudan, 2008).

Nevertheless, there are groups of commercial tools that execute security testing for SOAM, although they actually have several weaknesses and issues:

- Executing traditional security testing on SOAM technology may force the project to run over budget
- Traditional security testing does not work in parallel with the banking production environment
- Traditional security testing tools do not cover all SOAM layers

## Service Oriented Architecture Overview

SOAM is a group of related services and applications that communicate with each other to achieve certain advantages for enterprises. It is a client-server architecture which contains two major concepts services and consumers. SOAM has additional features to the traditional client-server approach such as a loose coupling feature between service components and separately standing interfaces (Natis, 2003). Additionally, SOAM architecture supports banks to innovate and maintain their business processes and strengthen their IT infrastructure (Early and Free, 2002; Bhuvaneswari and Jujatha, 2011).

### *SOA Architecture Layers*

In general, SOAM architecture contains the following major layers. (The number refer to items in Fig. 1).

### *Application and System Layer [1]*

This layer contains all the connected bank's e-services such as the internet, mobile banking, ECC, ATMs and others. These published systems serve the bank's clients through the internet (HTTPS secure URL) (Oracle, 2017).

### *Service Orchestration Layer [2]*

This layer is located in the middle of the SOAM architecture and is considered the source of functionalities containing all the common and unified bank's business rules and operations. For example, a request for transaction is a common function between internet banking, mobile banking and others, so it will be defined and upgraded and will then be used by all other e-services (Bhargavan *et al*., 2004; Lowis and Accorsi, 2009; Ye and Yang, 2013).

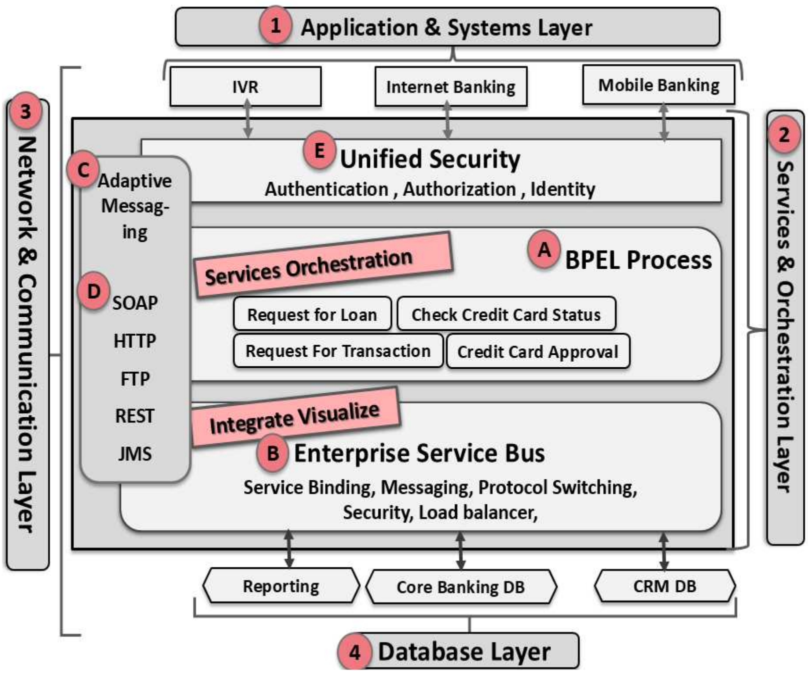

**Fig. 1:** SOA conceptual architecture

### *Network and Communication Layer [3]*

This layer works in parallel with all other layers that are transferring messages and binding services through SOAP or any other protocols.

### *Database Layer [4]*

This layer is considered as the source of truth. It is main purpose is to obtain all required information for the bank's e-services such as user transactions, client details, client account, credit card status, history and others.

### *SOA Architecture Components*

It is important to highlight the major components of SOAM conceptual design. These are as follows (The letters refer to items in Fig. 1).

### *Business Process Execution Language (BPEL) [A]*

BPEL is an XML based markup language that allows services within SOAM to interconnect, inter-operate and share both common business rules and data between core banking and e-services. It enables easy and flexible configuration for services into business processes and has recently become a major component in SOAM. BPEL is described as a WSDL (Oracle, 2017).

### *Enterprise Service Bus (ESB) [B]*

ESB is the core of SOAM architecture. It is considered as a single point of entry for all connected e-services and systems. Each system needs to communicate with ESB to influence all other e-services and each system is required to create special interface to interact and connect directly with ESB instead of creating several interfaces to communicate with many other systems (Oracle, 2017).

### *Adaptive Messaging [C]*

Adaptive messaging provides a communication pattern between orchestrated services and also between all clients and available e-services, such as requestor response, both synchronous and asynchronous. It helps in handling messages and manipulate them between different services through different protocols: SOAP, HTTP, REST and others (Oracle, 2017).

### *Simple Object Access Protocol (SOAP) [D]*

SOAP is an exchange messaging framework within different web services and systems. SOAP is the heart of web services schema, while web services are loosely coupled software components and standard methods for connecting web-based applications such as internet banking to the core-banking system by using XML, SOAP, WSDL and Universal Description, Discovery and Integration (UDDI) over Internet protocol backbone (Mainka *et al*., 2012):

- **XML:** For tagging the data
- **SOAP:** A message format for communication between systems involved in a web-service for transferring the data.
- **WSDL:** A mechanism for describing the services available
- **UDDI:** For listing what services are available within the architecture

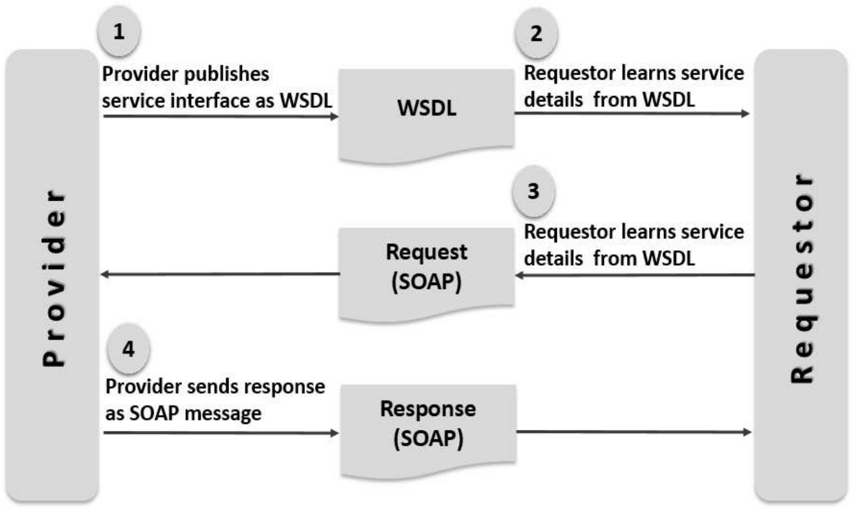
1
Provider publishes service interface as WSDL
WSDL
2
Requestor learns service details from WSDL
3
Requestor learns service details from WSDL
Request (SOAP)
4
Provider sends response as SOAP message
Response (SOAP)
Provider
Requestor

**Fig. 2:** SOAP protocol

SOAP is the master leader in communications. Its main purpose is to send and receive XML information to allow a successful communication between service requestor, service registry and provider. As shown in Fig. 2, SOAP is used to publish the descriptor into a service registry to:

- Send a service request from a requestor to the registry
- Send information from the registry to the requestor;
- Allow the requestor to bind to the service provider and run the web service (Mainka *et al*., 2012; Al-Jaroodi and Al-Dhaheri, 2011; Bhargavan *et al*., 2007)

It is important to mention that SOAP is essential for all web services and communication between banking e-services (Hariharan and Babu, 2014; Lux *et al*., 2005; Keen *et al*., 2017).

### *SOA Middleware Unified Security [E]*

SOAM unified security is responsible for establishing trust or handshaking relationship between two entities, such as core banking and mobile banking. In practice, in order to access secured e-services it must first provide information that express its origin of issuing or asserting. This process is referred to as making a claim, it may happen by providing credentials which are stored in a security token to allow claims to be asserted by sender authentication, to establish the identity service consumer by confirming this claim to be a true claim (Al-Jaroodi and Al-Dhaheri, 2011; Bhargavan *et al*., 2007):

- Authenticating means determining what the requester is allowed to do and authorization typically occurs after authentication
- Authorizing the consumer will require verifying the identity claim against predefined setup access rules
- Confidentiality is focused on protecting the privacy of information and making sure this information is available only to the authorized services when a message is being transmitted. A confidentiality mechanism is responsible for protecting the content throughout the message path
- Integrity is responsible for protecting the message from any alteration by unauthorized parties. The integrity mechanism ensures that the message remains un-attacked since its creation and transmission by the original sender (Bhargavan *et al.* 2007; Lowis and Accorsi, 2009; Keen *et al*., 2017)

## Motivation

This section discusses two major motivations which related to the current banking architecture, the available security testing tools and its problems and issues.

In Fig. 3 it is clear that all e-services are directly connected to core banking using web-services. These web services are built using different programming languages such as .NET, JAVA and others and different standard types, so it is more likely that an attacker will be able to modify or read the credential information for any web service or e-services.

Furthermore, web-services transmit data using SOAP. Unfortunately, this is not designed with a focus on security; so there is a possibility that the SOAP messages will be interfered during transmission. It is highly probable that an attacker or unauthorized user could act as the original sender then resend any critical communication to the original destination using network traffic. This scenario is called a replay, playback, rewrite attack, or XML rewriting attack. In web services it is possible for an attacker to record, modify, replay and redirect SOAP messages. There is a chance that attackers could inject malicious information which saved in XML files in order to modify or change business process and flow.

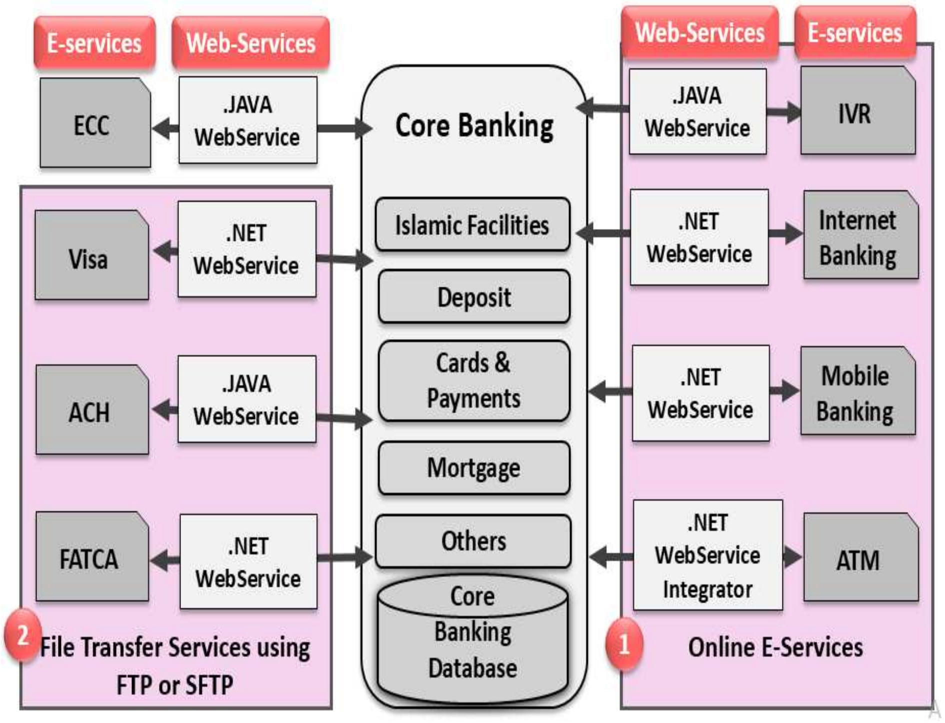

**Fig. 3:** Banking current architecture

Additionally, some banking e-services use a file transfer mechanism for transmitting data file between core banking and e-services or with the central bank, such as Visa, FATCA services and others. Unfortunately, this communication channel is not sufficiently secure and it is highly recommended that File Transfer Protocol (FTP) be converted to Secure File Transfer Protocol (SFTP).

In practice, to achieve a high level of security with this current architecture, it is strongly recommended that the following major points be considered:

1. Convert FTP into SFTP to allow transfer files within a secure channel
2. Group all file transfer services and unify them into one secure channel
3. Take into account that the lack of applied security standards for e-services means that it is easy for an attacker to modify or read the credential information for such services
4. Conduct full security testing to prevent the current architecture from different attacks
5. Conduct security testing with a third parties contract, this may cause any banking project to run over budget
6. Repeat the security testing after making several changes, applying upgrades or fixing bugs
7. Protect and keep scanning the network layer to avoid any vulnerabilities and attacks
8. Manage the network firewall in a secure way to monitor both income and outcome network traffic
9. Prevent the database from SQL injection attack
10. Increase the awareness of developers about how to write secure code over a possibly untrustworthy network
11. Consider how to mitigate the risk when converting current and traditional banking architecture into SOAM architecture
12. Consider how to have an automated security tool for the SOAM live banking environment without affecting performance
13. Follow a methodology to investigate the SOAM features and what are the most secure plugins that integrated successfully with SOAM architecture, also need to conduct several proof of concepts to check the implementation of SOAM with core banking, then finally propose a complete architecture with its related security analysis and evaluation

## Proposed SSOAM Architectural Design and Work Mechanism

The proposed work in this research provides a secure banking design architecture by implementing SOAM technology. It also provides an automated security framework working in parallel with the banking SOAM solution namely SSOAM.

The proposed SSOAM is an architectural design and a group of an integrated security plugins and tools. These plugins are responsible for scanning, finding, analyzing and fixing vulnerabilities which directly related to SOAM layers: The application and system layer; the service orchestration layer; the network and communication layer; and the database layer.

The SSOAM starts security scanning on an individual banking process to prevent both spoofing and denial of service attacks. After that, it is strongly recommended that the security of banking orchestrated processes and services be checked using ATLIST plugin. Then, since a high level of security will have been achieved, it is time to use TulaFale (Bhargavan *et al.*, 2004) to simulate playback attack on the network layer. Finally, it is necessary to prevent banking SOAM from SQLi injection, where an attacker is able to change banking business rules and workflow that are stored in the BPEL component while transferring SOAP messages and affect the database layer. This protection can be achieved using SOFIA Oracle plugin.

After running all these security plugins, it is important to run automated functional scripts to automatically double check the correctness of banking business rules within the BPEL component. It is also important to study and analyze vulnerabilities and attackers' behavior by using a business intelligence tool to predict and forecast new vulnerabilities and attacks in the near future.

As is clear from Fig. 4, SOAM technology is an integrated layer between core banking and e-services and the BPEL component stores all common banking business rules and processes using a unified programming language, in the case JAVA.

This proposed banking architecture mainly considered the following major points (The number refer to the items in Fig. 4):

1. No direct connection or communication is permitted between e-services and the core banking; any kind of communication must be through SOAM
2. Grouping all **file transfer services** into one channel will allow banks to focus on file security mechanism such as decryptions, encryptions, purging transfer instances and files, compression and decompression to achieve a high level of security on this channel. It will also be very easy to use SFTP instead of FTP
3. In the **file transfer section** in this scenario, Oracle Managed File Transfer 12c is a potential technology to be used for **file transfer services** to design an end to end secure transfer and this is successfully integrated with Oracle SOAM 12c. It has three sections, design, monitoring and administration and consolidates data file transfer into a centralized architecture to give complete visibility of all the file transfer mechanism. This architecture is clusterable, schedulable and available 24 h in 7 days per week
4. The proposed **automated security testing framework**, as mentioned previously contains a group of integrated security plugins. Each plugin should focus on one or more types of attacks to cover all SOAM layers and these plugins should work in sequence and in parallel with the production SOAM banking environment

Here are brief descriptions of these plugins:

### *[A] WS-Attacker*

One of the most important security tools to test an individual XML based web services (alone), this is a framework that uses SoapUI advanced API testing tool as the backend. This plugin is responsible for XML security especially XML signature which guarantees data integrity and authenticity within banking services. As shown in Fig. 5, this plugin works as below steps:

1. Load and scan an individual XML based web services (WSDL) by using SoapUI in the backend
2. Test the service by generating common attacks
3. Check the returned integer response from the SOAP
4. If the service is secure enough it will resist these generated attackers
5. Else, the web service needs to be improved and have a code review to fix the code related bugs from a security point of view
6. Return security testing result/report

The WS-attacker plugin was implemented in the proposed banking architecture (SSOAM) and the results has been showed that the proposed framework is secure.

Moreover, WS-attacker has been evaluated and tested by using a widely deployed web services frameworks such as Apache Axis2, JBossWS Native 6.0, JBossWS CXF 7.08 and .NET Web Services. The presented final results have been proved that WS-attacker can protect web services from SOAPAction, WS-Addressing spoofing, XML rewriting attacks and others refer to Table 1 (Hariharan and Babu, 2014; Mainka *et al.*, 2012).

### *[B] TLIST*

This plugin is to detect, analyze and manage vulnerabilities in banking orchestrated services and composite business processes. It also derives vulnerabilities types and pattern, which helps in preparation of any prevention actions.

In practice, this helps developers to write better and more secure code to avoid future vulnerabilities. Additionally, it tracks several types of attacks, such as those stopping and steering the business processes within SOAM standards, such as WSDL, SOAP and XML

(Lowis and Accorsi, 2011). It is important to note that ATLIST helps in checking both local and remote services and allows the checking of remote services without knowing their specific details, also helps in protecting PBEL business processes confidentiality and integrity.

Moreover, ATLIST plugin has been tested and evaluated then used for identifying the vulnerabilities in orchestrated business processes. This plugin has been developed for message alteration attack and the XML injection attack in WS-attacker.

ATLIST allows to check remote services without knowing the details about it, the below steps are related to its work mechanism:

1. Full scan for orchestrated services and business rules to find vulnerability and attack
2. Build and examine analysis tree
3. Use different kind of element as an input to build its analytical tree which help in vulnerabilities and attacks analysis
4. Refinement of vulnerability and attacks details
5. Return security test result / report

One of the most crucial attacks may face SOAM business processes is lost control of the services, when the attacker steer or stop the process and have the full control over it by message alteration attack, XML injection or SOAP injection by changing the header of transmitted message between services, ATLIST as a part of SSOAM architecture can protect banking services from SOAP injection attack and also many other attacks, refer to Fig. 6.

## *[C] TulaFale*

is a security testing plugin to verify the authentication properties of SOAP protocols, this is considered a specification language for defining a complete machine-checkable description of SOAP security protocols and checking BPEL business processes' authentication and help in verifying the authentication properties of SOAP, it has been tested and evaluated. (Bhargavan *et al*., 2004; Hariharan and Babu, 2014).

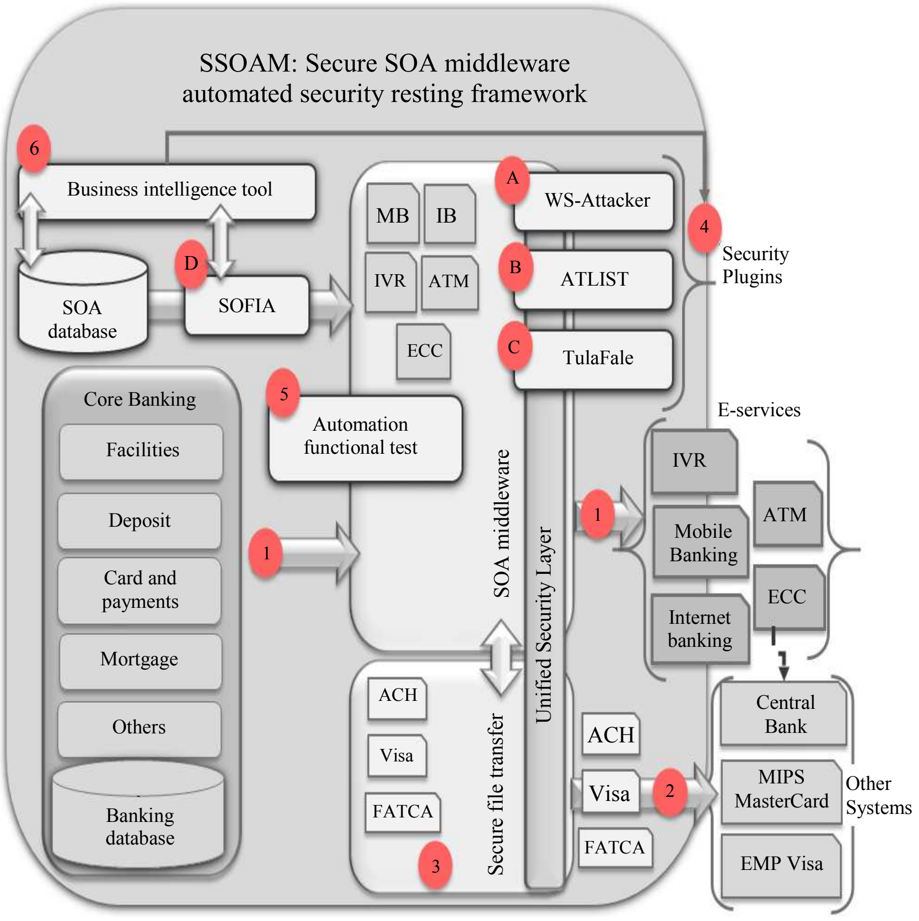

**Fig. 4:** SSOAM proposed architecture

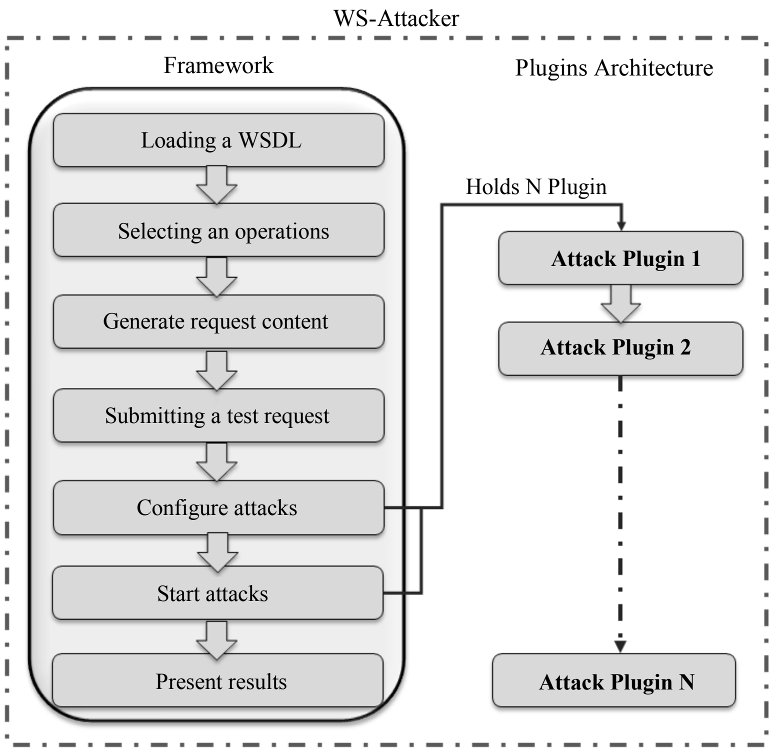

**Fig. 5:** WS-Attacker Work Mechanism (Mainka *et al*., 2012)

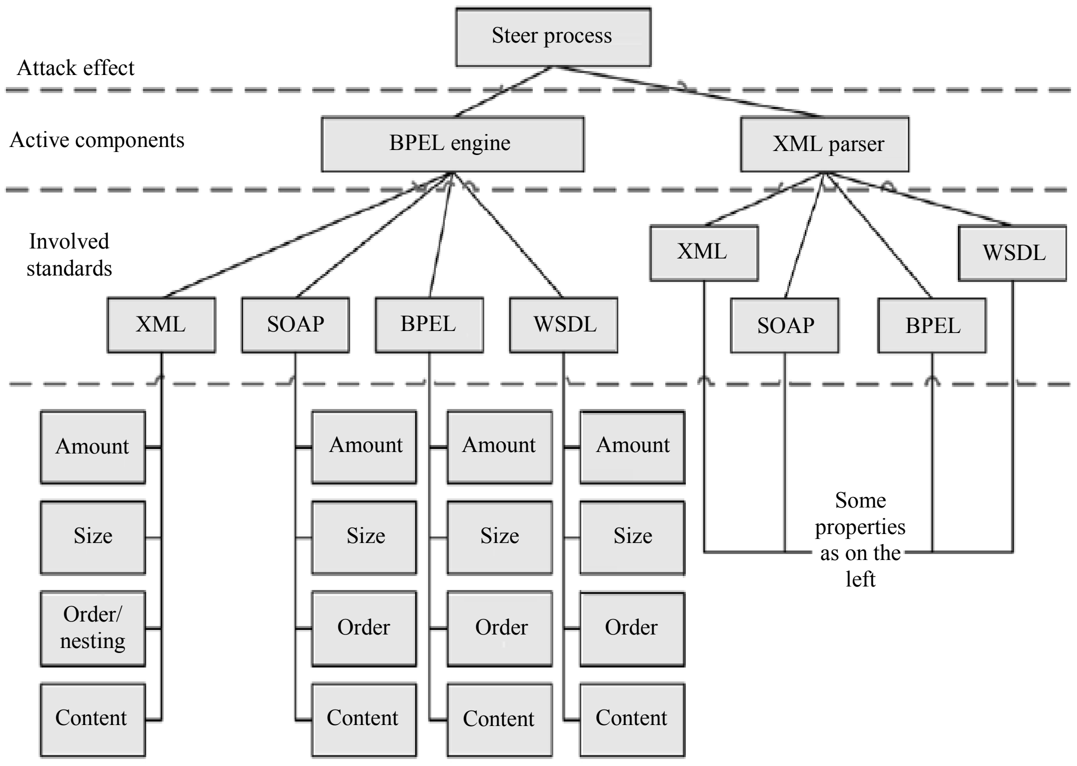

**Fig. 6:** Steering Process Attack (Lowis and Accorsi, 2011)

**Table 1:** Defeated attack per plugin

| # | Plugins | Defeated Attack | SOAM Layer |
|---|---|---|---|
| 1 | WS-Attacker | ✓ SOAPAction Spoofing<br>✓ WS-Addressing Spoofing<br>✓ XML-based DoS<br>✓ XML signature wrapping<br>✓ New Adaptive and Intelligent Denial-of-Service Attacks<br>✓ XML Encryption attacks<br>✓ Attacks on Symmetric Encryption Scheme. | Service layer |
| 2 | ATLIST | ✓ Prevents attackers from stopping, starting, splitting, spotting, steering and having any controls on banking business processes inside BPEL.<br>✓ Protects both service integrity and confidentiality. | Service orchestration layer |
| 3 | TulaFale | ✓ XML Re-writing attack.<br>✓ Web service session attack.<br>✓ SOAP traffic attack. | Network and communication layer |
| 4 | SOFIA Oracle | ✓ SQLi Attack | Database layer |
| 5 | Functional Testing | **Main Functionalities**<br>✓ Record functional test scripts and scenarios.<br>✓ Prepare both positive and negative test data to be used as an input for functional scenarios.<br>✓ Execute functional test cases<br>✓ Return security test result and report. | All SOAM layer |
| 6 | Business intelligence tool | **Main Functionalities**<br>✓ Study and analyze vulnerabilities and attacker's behavior.<br>✓ Predict and forecast new vulnerabilities and attacks. | All SOAM Layers |

Mainly, two types of standards are used:

- WS-Security Standard: Specifies a mechanism to secure SOAP protocol traffic only for one message a time
- WS-Secure Conversation: Specifies security contexts, to secure sessions between two web services or parties

The main advantage of using Tulafale library as a formal model is to make sure that the level of security for SOAP traffic within SSOAM architecture are guaranteed by using those standards WS-Trust and WS-Secure Conversation. Practically, Tulafale works as the below steps:

1. Check and verify the authentication properties of SOAP protocols and describe SOAP processors
2. Check the web services authentication
3. Return security test result/report

The level of security of authentication property within proposed SSOAM is more secure when implemented TulaFale tool.

### *[D] SOFIA the Security Oracles*

An automated security oracle that is integrated with several attack generation tools, to protect the banking database from SQL injection attack (SQLi). The accuracy of SOFIA oracle has been tested and evaluated successfully with three different attack generation tools on PHP and Java systems, no attack was missed and the rate of false positives was very low, which makes SOFIA a reliable and cost-effective approach.

As shown in Fig. 7, SOFIA work mechanism contains two major phases:

- Training: A group of legitimate SQLi are chosen from the safe model which contains all SQL executions log
- Testing: SOFIA oracles assesses tests new SQL statements then have a decision if this statement is legitimate or not

The main idea of using SOFIA is as a specific database firewall for production environment to help in filtering SQL statements and blocking any injection or attack. This can protect sensitive data from any insertion, alteration or deletion by unauthorized users and prevent such users from gaining control of the database server or performing arbitrary actions. SOFIA is used to catch real SQLi vulnerabilities with many inputs generated from attack-generation tools.

The proposed frame work was filtering SQL statement and blocking any SQL injection when SOFIA plugin is used in our approach. So, any amendment for data between entities will be detected and prevent any users from gaining control of the database server.

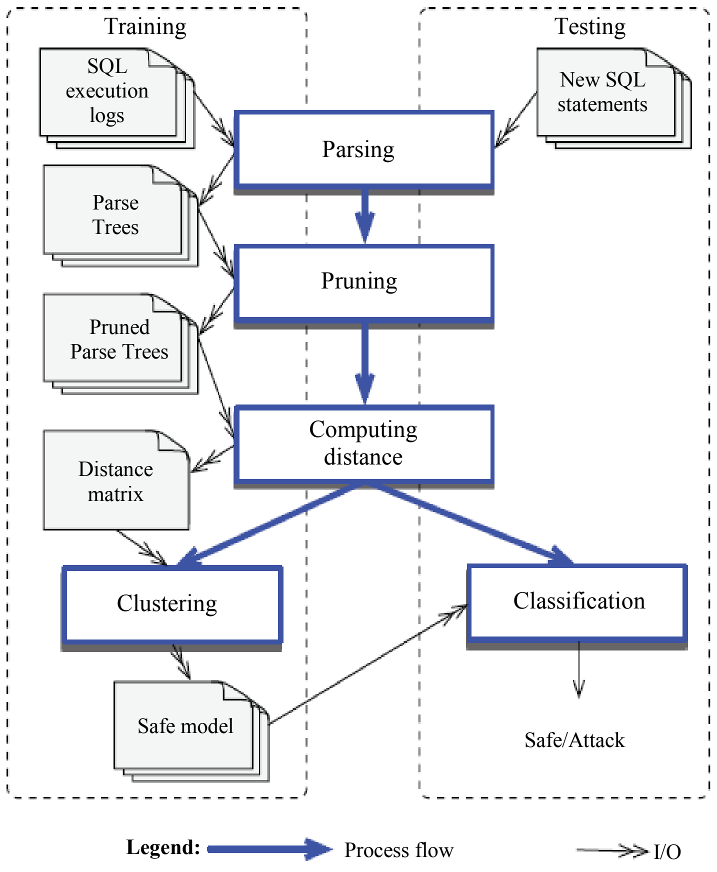

**Fig. 7:** SOFIA Oracles (Kajtazovic *et al*., 2017)

- **Functional automated testing tools** such as IBM Functional Tester, Selenium or any other potential automation tools, simulate all banking critical scenarios and business rules in an executable functional test case. These functional test cases must be executed dynamically when there is a possibility of finding an attack or facing risks, with the result of the execution either pass or fail. 'Pass' means that there is no change in the banking business rules, while 'fail' means that there is a change and more investigation is needed to double-check whether or not there is an attacker and what actions must be taken to mitigate or avoid this risk
- **Business intelligence tools** such as Tableau which are connected directly to both SOAM and security framework databases containing all historical transactions. This tool is used to study and analyze SOAM's attackers and vulnerability behaviors in previous years and forecast further behavior (Last, 2015)

## SSOAM Security Analysis and Evaluation

This section discusses the working mechanism of the proposed framework as well as analyses and evaluates its potential security features by using a banking business case then presented the defeated attacks which is summarized in the Table 1.

One of the common banking business scenario in SSOAM is related to get an approval for credit card transaction. It's worth to mention that such scenario will be securely protected within SSOAM framework.

The Banking business case has the following steps within SSOAM layers that already defined in Fig. 1 "SOA Conceptual Architecture" and in Fig. 4. SSOAM Proposed Architecture:

1 In the service layer the customer uses his credit card to start an online shopping transaction or point of sale mechanize
2 In SOAM unified security component, the customer's authentication, authorization and identity are approved. This is considered as the first line of defense

3 Then, all details are written within the customer module such as credit card, account number, expiry date etc. These details are written in SOAP format
4 This SOAP message is sent through the network layer from the web application to orchestrated services in BPEL processes
5 In the service orchestration layer the related process called Credit Card Approval unpacks the SOAP message and converts it into a command that the application can understand
6 The application gets the credit approval/ declined status in SOAP message after contacting the core-banking system through the service bus
7 This SOAP message is sent back to the customer
8 After unpacking the SOAP message, the customer gets the required information
9 Finally, all these steps are executed within a high level of security and protected from a crucial attack. Table 1 analyzes and summarizes what kind of attacks can be defeated when using SSOAM and on which specific layer

Additionally, in SSOAM there are potential points should be considered when implementing this security framework:

i. Each network packets should have two major countermeasures, namely timestamp and unique sequence number. This helps SOAM to accept or reject any packet received
ii. It is useful to generate a quality report indicating the severity and risk of vulnerabilities and the degree of impact in terms of how such vulnerabilities will damage or affect SOAM security (high, medium, or low)
iii. A business intelligence tool enables attackers' behavior to be studied and future vulnerabilities and attacks predicted
iv. This framework must run 24 h, 7 days per week, to check all messages communications between e-services and core banking and all packets through the network

Additionally, this SSOAM can prevent core-banking and harden its applications, BPEL business processes and database against Man-in-the-Middle (MITM) attacks. MITM is an unauthorized user sits between all parties' communications and tries to view and modify the most important messages between them. By having SSOAM, it will prevent core-banking and its services from steering services, also prevent from SOAPAction spoofing, which allows Man-in-the-middle attack to modify the 'SOAPAction' element in a HTTP header to execute an action that is not predefined by SOAM services, such as modifies access controls for banking business processes inside BPEL component. SOAPAction spoofing and MITM attacks can be mitigated by strictly verifying if the action in the SOAP body matches the action in the HTTP header. If they do not match, the incoming message will be rejected.

## Conclusion and Future Work

Achieving a high level of security in the banking domain is a challenging topic. This research discusses and promotes SSOAM, which contains two major ideas that should be applied in the banking domain to increase the security level. Firstly, at the level of SOAM architecture, it is highly recommended that all file transfer service mechanisms be grouped together and solid security features put in place to avoid any crucial attack. The recommendation is to use the Oracle managed file transfer product, which is an integrated component within Oracle SOAM 12c. Secondly, at the level of security testing, the study highlights the importance of having an automated security testing framework by grouping together different security plugins, automated functional testing and business intelligence tools to guarantee a high level of security for banking SOAM and then to use it in parallel with the production environment of SOAM core banking and its e-services 24 h, 7 days per week. This will also predict what kind of vulnerabilities and attacks may affect the banking SOAM.

The future work suggested by this research is to conduct a case study to validate the proposed secure architecture design in the banking domain; to validate the suggested automated security testing framework and the integration of security plugins, automated testing tools and business intelligence tools and to study and analyze any performance issues, also need to conduct performance and load testing for the SSOM by using IBM Rational Performance Tester.

## Acknowledgment

The authors would like to thank the reviewers for their valuable comments and suggestions that contributed to the improvement of this work.

## Author's Contributions

**Mustafa Al-Fayoumi:** Design the research plan and organized the study, coordinated the data-analysis, contributed to the writing of the manuscript and give final approval of the version to be submitted and any revised version.

**Ruba Haj Hamad:** Design the research plan and organized the study, coordinated the data-analysis, design architecture and contributed to the writing of the manuscript and revised version

**Jaafer Al-Saraireh:** Prepared the study and elaborate the methodology. Critical review of each version and correction.

## Ethics

This article is original and is not published in whole or in part elsewhere. There is no ethical issue involved in this article.

## References

Al-Jaroodi, J. and A. Al-Dhaheri, 2011. Security Issues of service-oriented middleware. IJCSNS Int. J. Comput. Sci. Netw. Security, 11: 153-160.

Bhargavan, K. and A.D. Gordon, 2017. Secure sessions for web services.

Bhargavan, K., C. Fournet, A.D. Gordon and R. Corin, 2007. Secure sessions for web services. ACM Trans. Inform. Syst. Security, 10: 8-8. DOI: 10.1145/1237500.1237504

Bhargavan, K., C. Fournet, A.D. Gordon and R. Pucella, 2004. TulaFale: A security tool for web services. Proceedings of the 2nd International Symposium on Formal Methods for Components and Objects, (MCO' 04) Springer, Berlin, Heidelberg, pp: 197-222. DOI: 10.1007/978-3-540-30101-1_9

Bhuvaneswari, N.S. and S. Jujatha, 2011. Integrating SOA and Web Services. 1st Edn., River Publishers, ISBN-10: 8792329659, pp: 330.

Ceccato, M., C.D. Nguyen, D. Appelt and L.C. Briand, 2016. SOFIA: An automated security oracle for black-box testing of SQL-injection vulnerabilities. Proceedings of the 31st IEEEACM International Conference on Automated Software Engineering, Sept. 03-07, ACM, Singapore, pp: 167-177. DOI: 10.1145/2970276.2970343

Early, A. and D. Free, 2002. SOA: A 'must have' for core banking (ID: SPA-17-9683).

Erl, T., 2009. SOA Design Patterns. 1st Edn., Prentice Hall, Upper Saddle River, ISBN-10: 0136135161, pp: 814.

Hariharan, C. and C. Babu, 2014. Security testing of orchestrated business processes in SOA. IEEE International Conference on Advanced Communications, Control and Computing Technologies, May 8-10, IEEE Xplore Press, Ramanathapuram, India, pp: 1426-30. DOI: 10.1109/ICACCCT.2014.7019337.

Inaganti, S. and S. Aravamudan. 2008. Testing a SOA application. BPTrends.

Kajtazovic, N., A. Höller, T. Rauter and C. Kreiner, 2017. A lightweight framework for testing safety-critical component-based systems on embedded targets.

Keen, M., R. Kaushik, K. Singh, B.A. Aghara and S. Simmons *et al*., 2017. Case study: SOA banking business pattern.

Last, D., 2015. Using Historical Software Vulnerability Data to Forecast Future Vulnerabilities. Proceedings of the Resilience Week, Aug. 18-20, IEEE Xplore Press, Philadelphia, PA, USA, pp: 1-7. DOI: 10.1109/RWEEK.2015.7287429

Lowis, L. and R. Accorsi, 2009. On a classification approach for SOA vulnerabilities. Proceedings of the 33rd Annual IEEE International Computer Software and Applications Conference, Jul. 20-24, IEEE Xplore Press, Seattle, WA, USA, pp: 439-44. DOI: 10.1109/COMPSAC.2009.173

Lowis, L. and R. Accorsi, 2011. Vulnerability analysis in SOA-based business processes. IEEE Trans. Services Comput., 4: 230-42. DOI: 10.1109/TSC.2010.37

Lux, K.D., M.J. May, N.L. Bhattad and C.A. Gunter, 2005. WSEmail: Secure internet messaging based on web services. Proceedings of the IEEE International Conference on Web Services, Jul. 11-15, IEEE Xplore Press, Orlando, FL, USA, pp: 75-82. DOI: 10.1109/ICWS.2005.138

Mainka, C., J. Somorovsky and J. Schwenk, 2012. Penetration testing tool for web services security. Proceedings of the IEEE 8th World Congress on Services, Jun. 24-29, IEEE Xplore Press, Honolulu, HI, USA, pp: 163-170. DOI: 10.1109/SERVICES.2012.7

Morris, E.J., W.B. Anderson, S. Balasubramanian, D.J. Carney and J. Morley, 2010. Testing in service-oriented environments.

Natis, Y.V., 2003. Service-oriented architecture scenario. Gartner Inc.

Oracle, 2017. Oracle SOA suite 12c – a detailed look.

Ye, Y. and C. Yang, 2013. Privacy protection for RBAC in service oriented architecture. Proceedings of the 26th IEEE Canadian Conference on Electrical and Computer Engineering, May 5-8, IEEE Xplore Press, Regina, SK, Canada, pp: 1-6. DOI: 10.1109/CCECE.2013.6567854