Service Oriented Computing and Applications
https://doi.org/10.1007/s11761-017-0221-1

ORIGINAL RESEARCH PAPER

Adaptive security architecture for protecting RESTful web services in enterprise computing environment

Mohamed Ibrahim Beer[1] · **Mohd Fadzil Hassan**[1]

Received: 30 April 2017 / Revised: 5 November 2017 / Accepted: 11 November 2017

Abstract
In this modern era of enterprise computing, the enterprise application integration (EAI) is a well-known industry-recognized architectural principle that is built based on loosely coupled application architecture, where service-oriented architecture (SOA) is the architectural pattern for the implementation of EAI, whose computational elements are called as "services." Though SOA can be implemented in a wide range of technologies, the web services implementation of SOA becomes the current selective choice due to its simplicity that works on basic Internet protocols. Web service technology defines several supporting protocols and specifications such as SOAP and WSDL for communication with client and server for data interchange. A new architectural paradigm has emerged in SOA in recent years called REpresentational State Transfer (REST) that is also used to integrate loosely coupled service components, named RESTful web services, by system integration consortiums. This SOA implementation does not possess adequate security solutions within it, and its security is completely dependent on network/transport layer security that is obsolete owing to latest web technologies such as Web 2.0 and its upgraded version, Web 3.0. Vendor security products have major implementation constraints such as they need secured organizational environment and breach to SOA specifications, hence introducing new vulnerabilities. Herein, we examine the security vulnerabilities of RESTful web services in the view of popular OWASP rating methodologies and analyze the gaps in the existing security solutions. We hence propose an adaptive security solution for REST that uses public key infrastructure techniques to enhance the security architecture. The proposed security architecture is constructed as an adaptive way-forward Internet-of-Things (IoT) friendly security solution that is comprised of three cyclic parts: learn, predict and prevent. A novel security component named "intelligent security engine" is introduced which learns the possible occurrences of security threats on SOA using artificial neural networks learning algorithms, then it predicts the potential attacks on SOA based on obtained results by the developed theoretical security model, and the written algorithms as part of security solution prevent the SOA attacks. This paper is written to present one of such algorithms to prevent SOA attacks on RESTful web services along the discussion on the obtained results of the conducted proof-of-concept on the real-time SOA environment. A comparison of the proposed system with other competing solutions demonstrates its superiority.

Keywords SOA · Web services · Security · REST · EAI

1 Introduction

Enterprise computing experiences paradigm shift when the integration of inter-related applications in an enterprise for achieving loose coupling of design, development, deployment and maintenance of small to large-scale computing systems becomes an obligatory requirement for survival and success in the current era of application integration that enforces enterprise application integration (EAI) standards. However, EAI has been facing many challenges due to the differences in platforms, database architectures, computing languages, and also with the different architectural solutions introduced by vendors. In addition, the legacy systems are no longer supported by original manufacturers [1]. EAI faces these compelling challenges in fulfilling three core purposes: (1) data integration across different systems, (2) independent

✉ Mohamed Ibrahim Beer
bmdibrahim@gmail.com

Mohd Fadzil Hassan
mfadzil_hassan@utp.edu.my

[1] Department of Computer & Information Sciences, Universiti Teknologi PETRONAS, 32610 Seri Iskandar, Perak Darul Ridzuan, Malaysia

Published online: 27 November 2017

of vendor architectures and (3) common facade for application integration.

Although EAI can be achieved using many different technologies, service-oriented architecture (SOA), which is based on loosely coupled independent business operational modules called "services," is the most recent strategic technological trend in the industry and becomes the selective choice by organizations owing to its simplicity. Web services are SOA implementations that work on basic Internet protocols such as Hyper Text Transfer Protocol (HTTP). There are two kinds of web services: Simple Object Access Protocol (SOAP) and REpresentational State Transfer (REST) [2,3].

The SOAP is an industry standard communication protocol that establishes stateful communication and provides data transport in the format of eXtensible Markup Language (XML) for web services as it uses service interfaces to expose business logic between the client and server. However REST is just an architectural style proposed by Roy Fielding in his Ph.D. thesis '*Architectural Styles and the Design of Network-based Software Architectures*' [4]. The REST provisions stateless data transfer using any web-supported data formats such JavaScript Object Notation (JSON) and XML between the client and server that works based on direct HTTP methods such as GET, PUT, DELETE and POST through URI. Exposing business services based on this REST principle are referred as RESTful web services.

As REST is lightweight in communication, it brings a good number of advantages to enterprise computing such as less bandwidth and increased speed for services. However, REST is not a protocol and it has no meta-descriptions due to the absence of service interface. This has resulted in the unavailability of specific security model unlike the one established in SOAP web services. At present, the security for REST data transfer is dependent on network and transport layer securities that are totally obsolete due to the latest generations of network communications and their vulnerabilities [5–10] such as transport layer security (TLS) channels that should be frequently reset for non-point-to-point communications.

The existing and relevant security solutions for REST security are analyzed as part of literature review and found that none of them result in a comprehensive security solution for the entire RESTful web services design without violating REST and SOA principles. The researchers proposed few security solutions; however, they may result in introducing additional vulnerabilities as usually security protocols are interpreted to be tricky, while the vendor security products operate mostly on trusted and secured organizational environment that needs specific products installed.

In this paper, we are proposing an adaptive security architecture enforcing EAI security on protecting RESTful web services from SOA attacks in both secured and non-secured enterprise computing environments without breaching any REST and SOA principles. The conducted research highlights that RESTful web services on this proposed security architecture can be a better alternative to the currently popular SOAP-based web services in the industry.

The remaining sections of this paper is organized as follows: Sect. 2 reviews the research topic in terms of validating its importance. The methodology and functionality of the proposed security architecture is illustrated in Sect. 3. The implementation and validation analysis are discussed in Sect. 4. The related works along with a discussion and comparison with our contributions are given in Sect. 5. Section 6 concludes the paper by showing original contributions, limitations, impacts and potentials for future development.

2 Context and motivation

The following subsections describe the architectural model of REST for web services, its need of security with real-time statistics on security vulnerabilities, the motivation behind conducted research and a review on the available security solutions for REST in respect of the literatures.

2.1 The skeleton of RESTful web services

The RESTful web services does not maintain a separate registry to persist the service description details unlike SOAP-based web services do. The basic architecture of SOAP-based web services is depicted in Fig. 1. The service descriptions are published into Service Registry in the form of Web Services Description Language (WSDL) by service provider, and then, the service requester invokes the required service from the service provider based on the defined service interface through SOAP protocol. Security can be enforced through metadata descriptions on the SOAP request/responses. It is not in the case of RESTful web services that the service requester directly calls the Service Provider through HTTP methods, as outlined in Fig. 2. In this REST architectural principle, applying security on the metadata elements of underlying web service call is not possible. Hence, RESTful web services are more prone for attacks than SOAP-based services.

2.2 Security vulnerabilities on RESTful web services

The Open Web Application Security Project (OWASP) is an international nonprofit organization that deals with web application security [11]. It collects and observes information related to security incidents at a global level periodically and provides a list of the top vulnerabilities of web application security; its report on 'Top 10 Vulnerabilities' is well recog-

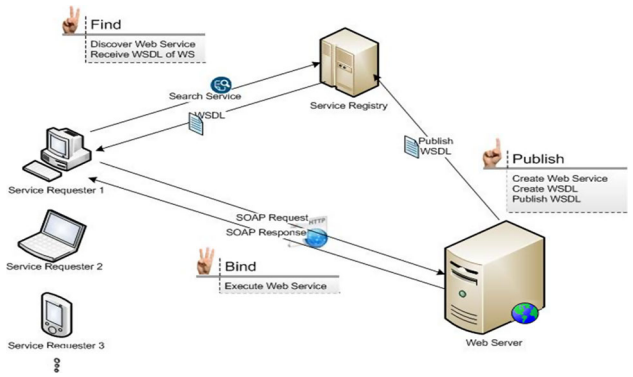

Fig. 1 SOAP-based web services

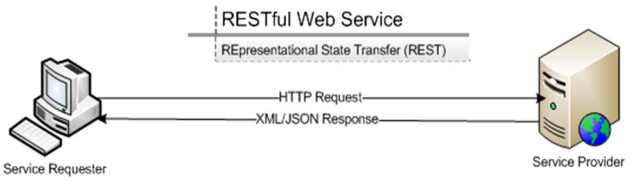

Fig. 2 REST -based web services

nized in the security industry and followed up by prominent vendors of security products. As a RESTful web service is basically a web application, the following security vulnerabilities listed by OWASP's rating methodology should be dealt with thoroughly to provide secure services.

- Injection attacks and message altering
- Authentication-based attacks
- Denial of service (DoS) and buffer overflows
- Cross-site scripting/cross-site request forgery
- Man-in-the-middle (MITM) attacks
- Replay attacks and spoofing
- Insecure direct object references
- Sensitive data exposure
- Missing function level access control
- Unvalidated redirects and forwards
- Malware malfunctions

As REST API uses simple Internet-based HTTP/HTTPS protocols instead of using complex application-specific protocols such as CORBA, COM/DCOM and RMI, it is prone to all application layer security vulnerabilities such as XSS and parameter tampering [12–14]. Generally REST deals with JSON format for data exchange; it is possible to subvert or disturb the application logic on the services by JSON injections, which may result in misbehavior of services, data theft, resource deletion and malware execution. As all the web service-based interactions are performed over the web with text based data exchange due to HTTP's nature, certainly there is a lack of security proof for protecting data for web services implementation of SOA.

The 2015 "Cost of Cyber Crime" study by HP Enterprise Security [15] states that the average annual cost of cybercrime is USD 15 million, ranging from USD 1.999 to USD 65 million each year per company in USA. According to a report from Symantec [16], over 431 million new malware samples were exposed in 2015, meaning about one million threats are being released daily. The Web Applications Security Statistics Report, published in 2016 by WhiteHat Security, Inc. [17] states that although application security solutions have been available for years, vulnerabilities remain rampant; developers and IT team need to prioritize security as much as functionality when developing, customizing or implementing applications. The percentage of occurring denial of service (DoS)/distributed denial of service (DDoS)-based attacks were not getting reduced over the years even though new hardware-based solutions are introduced [18,19]. Security aspects must be considered thoroughly at the time of designing, because web services require high security due to its architectural design.

2.3 Limitations in current RESTful security

Out of the available security solutions in the industry for securing web based transactions, two algorithms namely OAuth [20] and OpenID [21] which are widely applied for REST security in many organizations. However, these two solutions are not intentionally developed for securing RESTful web services and also cannot claim to be as the algorithms used on these two solutions are dependent on user session where no user session should be maintained as per REST principle. The OpenID uses session token for the transaction between client and web server, obviously these kinds are tokens can be easily interpreted by man-in-the-middle attacks. The OAuth works based on trust approved by third party that the service requester should get authorized by authorization server first so that it can consume the projected resources at service provider for the defined duration as per the agreed-on policy between resource owner and authorization server on accessing resources. In general, this OAuth is considered to provide clients with a "secured delegated access" to server resources on behalf of the resource owner. However, there are possibilities that this algorithm may result in additional threats to SOA, as listed below [21–23].

Potential threats to service requester include the following:

- Interception on the data transfer with the service requester and authorization server, targeting to obtain credentials and session details.

- Interception of data communication between the service requester and protected resource, aimed to acquire sensitive business data.
- Spoofing attacks on the protected resources of the resource owner by obtaining access token that belongs to a valid service requester.
- Redirecting the service requests to a web service which is monitored and controlled by an attacker.

Potential threats to the authorization server include the following:

- Interception of client's credentials and authorization codes/tokens aimed to do replay attacks.
- Brute-force attempts to disclose confidential data of the service requester and/or refresh access tokens.
- Interception-based attacks to redirect response Uniform Resource Identifiers (URIs) for leading access tokens to be sent to attackers.
- Cross-site request forgery attacks which abuse legitimate service requester sessions with service provider for attacker-controlled clients; also results in denial of service attacks.

Potential threats to the protected resource include the following:

- Replay for authorized requests but the requests are forgery.
- Attempt to access to unauthorized resources which are available in the same service provider while accessing authorized resource by valid token.
- Brute-force attacks on guessing for access tokens and making stress the service provider on accessing a sensitive resource.
- Presentation of intercepted authorization codes by unauthorized clients.

Potential threats to the resource owner include the following:

- Interception of resource owner credentials and misuse.
- Delegation of excessive access to resources on the service provider.
- Compromised clients attempt to abuse delegated access to manipulate data.

3 The proposed security solution

The following subsections describe the proposed security architecture, security algorithms and their implementation in enterprise computing applications using AOP.

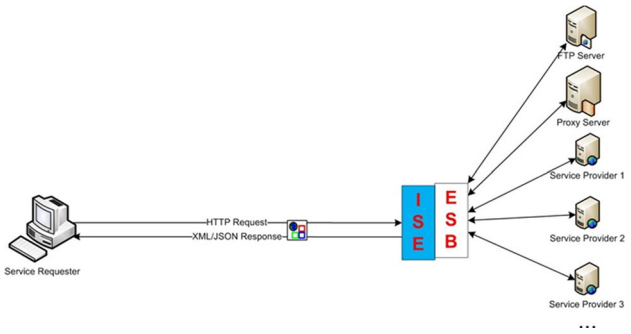

Fig. 3 Proposed security engine placed on RESTful web service design

3.1 Intelligent security engine: the security component

The proposed security architecture is developed on the basis of public key infrastructure (PKI) and its related technologies to provide protection for RESTful web services defending against EAI attacks through message integrity, confidentiality, authentication, and single stateless request authorization which are implemented and governed by a pluggable novel component named "intelligent security engine (ISE)" attached to SOA architecture at a server, proxy, enterprise service bus (ESB) network or even on any computing node without disturbing the source code of the web services, an example of attaching ISE to an ESB network is shown in Fig. 3.

This ISE follows Interception SOA Design Pattern and works based on rule-based engine (RBE) which supports fraud detection on the service requests/responses, and the 'intelligent' part of this component incorporates artificial neural networks (ANN) learning techniques for supervised knowledge acquisition process on detecting security threats of SOA in secured and in-secured enterprise computing environment. The design of ISE on securing SOAP-based web services and its implementation are presented in the paper [24] by the same authors that the security solution is constructed in an adaptive way forward of predicting the security vulnerabilities for the underlying EAI environment, preventing them from occurrence, and learning for identification of potential threats, as a cyclic process shown in Fig. 4.

Adaptive security architecture is defined as one of the top 10 strategic technology trends for 2016 in the industry [25] provides flexibility on security measures for the protection that goes beyond the traditional perimeter defense mechanisms.

This ISE is further enhanced with PKI infrastructure for supporting REST-based web services. This security component is designed using Aspect-Oriented Programming (AOP) concept which decouples the security solution from the underlying business logic of the services. AOP supports

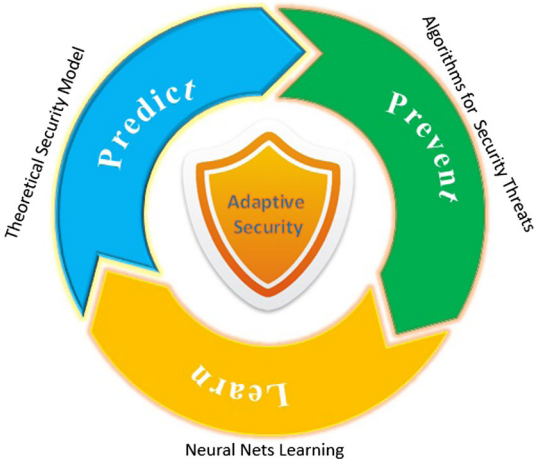

Fig. 4 Proposed adaptive security architecture

modularity of software systems by encapsulating crosscutting concerns. Crosscutting concerns generally refer to non-functional properties of Software such as security, synchronization, logging, which are tangled and scattered across the application code. In AOP terms, the joinpoint is a point in the control flow logic of the software application such as method call, object construction, or field access. A pointcut defines at what joinpoints, the associated "Advice" should be applied. An advice is a code fragment executed when joinpoints satisfying its pointcut are reached; all the required AOP entities can be configured using XML outside the underlying SOA applications. Hence, the proposed security solution acts as a 'pluggable' component which needs no or minimum code change while attaching it to SOA applications, as illustrated in Fig. 5.

3.2 Proposed security algorithms

This proposed security architecture expects that the security algorithms given in Figs. 6 and 7 should be applied at both the client and server ends, respectively. All the given steps in the algorithms are self-explanatory and use common cryptographic methodologies such as XML Key Management Specification (XKMS), symmetric/ asymmetric keys, and hashing techniques along with the REST terminologies.

The algorithms utilize trusted PKI setup to retrieve private/public keys and certificates using client/server identifiers so that exchanging the keys/certificates within the message itself for client/server interaction becomes not mandatory. This arrangement not only reduces the message payload, but also ensures that only the right node obtains the keys and certificates.

The required metadata are incorporated in request/ response headers, not as part of message payload. This is one of the significant features that makes the services to do not disturb the existing structure of REST protocol and significantly increases the speed of security metadata parsing when compared with other similar security proposals that were developed for SOAP-based web services such as WS-security.

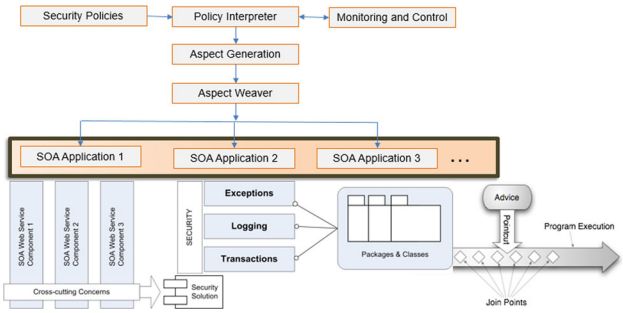

Fig. 5 Security is plugged-in using AOP for SOA

To distinguish the added metadata attributes from the normal HTTP header attributes, "ISE_" is prefixed to the newly introduced attributes. This setup is not mandatory for the proposed algorithms to run; however, it is highly recommended for readability and tracking. Also the chosen cryptographic techniques can be parameterized/configured at global level or message level according to the organizational security policies. To align with the defined REST response Service Level Agreement (SLA), these algorithms are written to use symmetric encryption for handling larger message payloads instead of asymmetric keys where the asymmetric cryptography takes more time for decryption process when compared to the symmetric one.

3.3 Followed-up security patterns

The proposed solution comprises the following core security patterns along with cryptographic solutions for REST security.

- *Request filtration* Monitor and filter for any malicious data which are coming from untrusted user inputs in JSON/XML format.
- *Authenticate clients* Monitor and filter the URI for any request coming from malicious clients.
- *Protect services* Protect the entire surface of the REST services, defending against security vulnerabilities listed by OWASP on web applications.
- *Authorize API services* Protects XML/JSON parsers against injection level attacks, malicious code attacks and buffer overflow attacks.
- *Service scalability* The proposed security solution does not affect the horizontal and vertical scalability of the services.
- *Service auditing and analysis* As the security code is decoupled from the actual business logic code, auditing

```
/* Applied at Web service client side */
Algorithm: REST_Protection_At_Client_Side
Input:      request: RESTful Service Request
Variables:  hdr: REST header request
            msg: REST request body
            digAlgm: Chosen hash algorithm name
            digestValue: Fingerprint  of request
            url: Full path of REST request URI
            clientId: Client's unique id in PKI
            serverId: Server's unique id in PKI
            clientCertId: Id of client's certificate in XKMS
            clientPriK: Private key of client
            sign: REST Signature
            symK: Symmetric Key
Output:     requstURI: REST request URI to access the service
Begin
Step I:     /* Access header and body of request */
            hdr = request.Header
            msg = request.Body
            Verify msg on JSON format
Step II:    /* Derive fingerprint of msg */
            digestValue = Digest [msg.payload]digAlgm
            hdr['ISE_HASH_ALGM'] = digestValue
            url = request.uri.absolutePath
Step III:   /* Construct REST Signature */
            clientCertId = PKI.certId[clientId]
            clientPriK = XKMS[clientCertId].privateKey
            valueForSign = (digestValue + url + clientCertId + hdr)
            sign = Encrypt [valueForSign]clientPriK
            /* Assign the sign to Request Header */
            hdr['ISE_REST_SIGN'] = sign
Step IV:    /* Encrypt the message if configured */
            // Configuration can be obtained from Global properties
            // file for request; it is specified in the service for response
            If Encrypt_Request is enabled Then
              symK = Call GenerateSymK(symKAlgm)
              msg.payload = Encrypt [msg.payload]symK
              For Each attribute in hdr
              Do
                hdr[attribute] = Encrypt [hdr[attribute].value]symK
              Done
              hdr['ISE_REQUEST_ENCRYPTED'] = True
            End If
Step V:     /* Secure the symmetric key */
            serverPubK = XKMS[PKI.certId[serverId]].publicKey
            encSymK = Encrypt [symK]serverPubK
            hdr['ISE_KEY'] = encSymK
Step VI:    /* Assemble the REST URI */
            requstURI.Header = hdr
            requestURI.Body = msg
            Redirect the request to requstURI and wait for response as
            REST works on Synchronous mode
            response = response from RESTful web service
            /* Execute the following steps specified in the server-side
            algorithm */
            Obtain the symmetric key
            Do decrypt the message if configured
            Evaluate the digest from client
            Ensure message integrity
            Check for any man-in-middle attacks on IP
            Check for valid client
            /* Parse the message */
            Parse the response in the given data format and pass the
            data to the client module for processing
End
```

Fig. 6 RESTful web service protection (client side)

```
/* Applied at Web Service server side */
Algorithm: REST_Protection_At_Server_Side
Input:      requstURI: RESTful Service Request
Variables:  hdr: REST header request
            msg: REST request body
            digAlgm: Chosen hash algorithm name
            clientDigestValue: fingerprint of request
            url: Full path of REST request URI
            clientId: Client's unique id in PKI
            serverId: Server's unique id in PKI
            clientCertId: Id of client's certificate in XKMS
            serverPriK: Private key of server
            clientSign: REST Signature from client
            symK: Symmetric Key
Output:     response: Response to the client
Begin
Step I:     /* Interception of request by Security Engine [ISE] */
            request = request[requstURI]
            hdr = request.Header
            msg = request.Body
            Verify msg on JSON format
            If request.Method in ('GET' , 'POST', 'PUT') Then
               Go to Step II
            Else //such as 'DELETE'
              Authenticate and Perform the requested HTTP action
            End If
Step II:    /* Obtain the symmetric key */
            encSymK = hdr['ISE_KEY']
            serverPriK = XKMS[PKI.certId[serverId]].privateKey
            symK = Decrypt [encSymK]serverPriK
Step III:   /* Decrypt the message if configured */
            If hdr['ISE_REQUEST_ENCRYPTED'] = True
            Then
              msg.payload = Decrypt [msg.payload]symK
              For Each attribute in hdr
              Do
                hdr[attribute] = Decrypt  [hdr[attribute].value]symK
              Done
            End If
Step IV:    /* Evaluate the digest from client */
            clientSign = hdr['ISE_REST_SIGN'].value
            clientId = Map client IP Address to Client Id
            clientCertId = PKI.certId[clientId]
            clientPubK = XKMS[clientCertId].publicKey
            valueOfClientSign = Decrypt [clientSign]clientPubK
            clientDigestValue = valueOfClientSign.split[0]
            digAlgm =
            valueOfClientSign.split[3]['ISE_HASH_ALGM']
Step V:     /* Ensure message integrity */
            calculatedDigest =  Digest [msg.payload]digAlgm
            If calculatedDigest <> clientDigestValue Then
              Throw Error ('Message is altered')
              Handle Error and Audit; End.
            End If
            /* check for any man-in-middle attacks on the IP */
            If requstURI.IPAddress <>
              valueOfClientSign.split[1].IPAddress  Then
              Throw Error ('IP Address is altered')
              Handle Error and Audit; End.
            Else
              Record IP address and time of request
            End If
            /* Check for valid client */
            If clientCertId <> valueOfClientSign.split[2] Then
              Throw Error ('Client certificate error')
              Handle Error and Audit; End.
            End If
```

Fig. 7 RESTful web service protection (server side)

```
Step VI:    /* Prevent DDoS attacks */
            currentRequestTimestamp =
                        requestURI.receivedDateTime
            startTimestamp = currentRequestTimestamp –
                             DefinedDeltaTime
            Increment 'requestCount' for each received request
            If requestCount in
              [startTimestamp, currentRequestTimestamp]
               > MaxAllowedRequestThreshold Then

              clientRequestCount = requestCount with
                        requstURI.IPAddress matched
              If clientRequestCount >  ThresholdByIP Then
                  Throw Error ('DoS Attempt detected')
                  Block requstURI.IPAddress
                  Handle Error and Audit; End.
              End If
            End If

Step VII:   /* Process the request and obtain the response */
            Forward the request to the corresponding service.
            response = call service(request)

            If response.error.count <> 0 Then
               Construct error response
               Put error indicator on message header
               Audit; End.
            End If

Step VIII:  /* Follow the steps specified in client side algorithm */
            Derive fingerprint of response.Body
            Construct REST Signature
            Do encrypt the message if configured
            Secure the symmetric key
            Assemble REST response

            Forward the REST response to the client

End
```

Fig. 7 continued

can be easily applied on the consumption of services and analyzed accordingly.

- *Security constraints and access control* All the modern web environments allow verb-based authentication and access control (VBAAC), where the REST applications are tied up to these HTTP methods, aka verbs for its create, read, update and delete (CRUD) operations. These verbs should be properly interpreted as which methods are allowed for what resources due to the constraint that all the verbs are not valid for every resource available in the service provider.

4 Validation and discussion

The proposed security solution is developed and tested with large-scale financial data in a real-time SOA environment that implements both RESTful and SOAP web services. The hardware and software configuration that is used for this proof-of-concept (PoC) is as stated in Table 1.

A sample request/response before and after applying the proposed security solution is given in Figs. 8 and 9, respectively. The request and response message payloads were encrypted after the proposed security solution was applied to a RESTful web service. This encryption avoids man-in-the-middle attacks, message eavesdropping, data tampering, and replay attacks. The actual request URI along with client's IP address is included in the construction of REST signature at the client end, and it is verified against the received IP address of the client at the server side. This client IP validation check prevents IP spoofing attacks and IP router manipulation attacks.

The cryptographic keys are encrypted and shared using a closed PKI infrastructure within the organizational boundary; this prevents dictionary and brute-force attacks. Because the body of the request/response of RESTful web service is checked for predefined validation, the proposed algorithm will prevent recursive payloads, oversized payloads, and schema poisoning attacks. A dedicated interceptor in the algorithm which monitors all the requests and responses, and hence prevents DoS/DDoS attacks. At present, the proposed algorithm is not written to concentrate on preventing code level attacks such as XPath injection, rewriting attacks, access control and attacks caused by improper code. However, the security solutions for these attacks can be developed separately and attached to this proposed architecture as it is constructed as an adaptive specification, not as a security product.

The obtained results on prevention of DoS/DDoS attacks with the available web security, vendor product and proposed security solution are represented graphically in Fig. 10. The critical analysis on the results shows that the proposed security solution is been stable for preventing DoS/DDoS attacks at long run where the competing vendor-based solution (hardware-based solution that works on IP router level) cannot be sustained when the number of DoS requests is increased and the defaulted web security is totally inadequate for protecting RESTful web services. In the other side, the proposed algorithm consumes more processer power than the IP router-based vendor solution as charted in Fig. 11 and reaches the maximum utilization earlier than vendor-based competing solution. However, the varying delta time that is absorbed for security-centric applications is considerable as security is the major concern.

Figure 12 illustrates the overall view of message-level attacks on RESTful web services and their protection using the defaulted network/transport layer web security, competing security solution—OAuth authorization framework, and proposed security solution using PKI. The results show that the defaulted basic security is totally inadequate for REST security, and the OAuth performs well for protecting REST messages based on authentication concerns of the attacks.

However, the overall protection by OAuth framework becomes saturated after the number of requests increased past a certain level, where the proposed security method performs better in this case and provides stable protection. However, it is very difficult to achieve a comprehensive security solution

Table 1 SOA testing environment for PoC

Hardware/software	Configuration/capacity
Processor	Intel(R) Xeon(R) @ 2.30GHz; 16 CPUs; 64 bit
Total main memory	32 GB; secondary storage: 500 GB
Operating system	Red Hat Linux 4.8
Application server	Tomcat 7
Web services	JAX-RS for REST; JAX-WS for SOAP based; developed using spring framework (Java) and restlet framework
Crypto algorithms	Asymmetric keys: RSA; symmetric key: blowfish hash function: MD5; XKMS: open XKMS

that provides 100% security using only software components. At the same time, the proposed adaptive security architecture to SOA certainly increased the security ratio for RESTful web services even compared to the available security solutions for SOAP-based web services.

5 Comparison of proposed solution with related works

Constructing security solutions for SOA, particularly for RESTful web services, is a new and active topic in enterprise computing research. This section reviews the existing related works at high level and attempts to compare the proposed security solution with those related works.

Neha et al. [26] presented a security solution to prevent distributed DoS (DDoS) attacks on RESTful web services by validating the number of request URIs and IP addresses from which the requests were made. Lee and Mayur [27] performed a critical analysis of JSON input attacks and provided an integrated solution to defend JSON input against these attacks using static source code analysis and input validation techniques. Sungchul et al. [28] developed an identification-based authentication algorithm to achieve secure RESTful web service access. In addition, Orellana et al. [29] considered asynchronous distributed computing and on-demand computing using RESTful web services that use Apache security modules to protect data transfer.

Serme et al. [30] found that the current ad hoc security mechanisms for RESTful web services are error-prone when implemented for secured data interchange and transport layer security offers poor flexibility. They introduced a new security protocol using the REST signature method and analyzed the result. They claim that their proposed method is better than the existing transport layer/ad hoc-based security solution for REST web services; however, it can be applied only for independent and single services and not for a group of services. Sudhakar [31] analyzed how REST security differs from HTTP security and suggested various security mechanisms to address REST security challenges. Moreover, he proposed a token-based security model for securing RESTful web services. Similar kinds of token-based security solutions were also proposed by Malisetti [32] and Adamczyk et al. [33].

Request:

```
POST /container/list HTTP/1.1
Accept: application/json
Content-Type: application/json
Host: 192.160.1.23:8080
Connection: keep-alive

{
 "accountDetails":[
                   {"accountNumber":"136007345623"},
                   {"accountType":"SB"}
                  ],
 "fromDate":"20170401000000",
 "toDate":"20170414235959"
}
```

Response:

```
HTTP/1.1 200 OK
Server: Restlet-Framework/2.2.3
Content-Length: 1094
Content-Type: application/json; charset=UTF-8
Date: Mon, 17 Apr 2016 15:42:19 GMT

{
  "txnList": [
    {
      "txnNumber":"SBT3928738456207",
      "txnDateTime":"20170403105534",
      "type":"D",
      "amount":"1500.00",
      "ccy":"MYR",
      "payeeId":"32984897654",
      ...
    },
    ...
  ]
}
```

Fig. 8 Sample request/response before proposed security solution is applied

Brachmann et al. [34] proposed a central security service with a lean API that handles both authentication and authorization for trusted RESTful services; provides role-based access control for accessing the services. Furthermore, possible security solutions for RESTful web services using OAuth authentication were analyzed and proposed [20,35,36].

The existing security models offer IP-to-IP-based security solutions, but the expected one is application-to-application-based security solution. Popular security mechanisms such

Request:

```
POST /container/list HTTP/1.1
Accept: application/json
Content-Type: application/json
Host: 192.160.1.23:8080
Connection: keep-alive
ISE REST SIGN: lZ90s8iAT/MQLzLVOczUcLkaXRIIWlE3Wx5f5r1v
               f1GU3s2lAiAN/p6jYRXTjqPCQYklpu9HUQsQ==
ISE_REQUEST_ENCRYPTED: True
ISE_KEY: SaF3MeV0p4Ov6BixSlVV8Zx5DgwU1PwWBDO8QXqgQz
         HBUdPEZekDFnJpQAh12D6k4PeffmkjQJkg==

53274AB5FCA383A82F5A55CB57C580FFDA7516D573532E5BA038939A
B62D3F11092280111968 46B528FA9964238CAAB94732B55C75BEB92A
61D64467504DDD0DA038939AB62D3F11B88E9BCE8E62A66333B6A42
...
```

Response:

```
HTTP/1.1 200 OK
Server: Restlet-Framework/2.2.3
Content-Length: 1853
Content-Type: application/json; charset=UTF-8
Date: Mon, 17 Apr 2016 16:05:14 GMT
ISE REST SIGN: THiv8/Aaq2pLuOSmbOqcOGiaDTIPFXWvaxMVoVRCF
               Phkfgm0ENSjIgPeFID5d8VGcKCtjELcfmlw==
ISE_RESPONSE_ENCRYPTED: True
ISE_KEY: SaF3MeV0p4Ov6BixSlVV8Zx5DgwU1PwWBDO8QXqgQz
         HBUdPEZekDFnJpQAh12D6k4PeffmkjQJkg==

99BDF49DEDCD4571B1DA170CB5137DD41169B8D66DAD89A7454B077A
B583ECFB963C4B6F877EE2BA4E7A6BFBB0ECEC4F2014640DE3E5E127
5FCC4DEC6C67BFF7342F86D7DCBD0D9FAE620F6B49A45088A71B800
...
```

Fig. 9 Sample request/response after proposed security solution is applied

as OAuth and OpenID violate the core principles of the REST that these algorithms work based on session, but REST does not maintain any session. Therefore, there is an urgent need of having an adaptive security architecture extending HTTP security models for protecting RESTful web services in the enterprise computing environment, respecting REST principle. In this case, our proposed security solution fills in the gaps where the listed related works do not able to achieve.

As the proposed security algorithms are using the existing benchmarked security standards such as PKI and hashing techniques, the computational overheads for their implementation are minimum and the conducted PoC results show that average execution time for these proposed algorithms take less than 1 s on 90% of the test cases, where this 90% test analysis is the industrial standard benchmarking cut-over to accept the obtained performance is good. The proposed system's precision for security protection in face of other network attacks such as dictionary attacks, man-in-the-middle attacks, replay attacks, session management attacks and cross-site scripting attacks is observed from the PoC results and depicted in Fig. 13. This test is conducted on the underlying five different SOA attacks with default OSI network layer security, vendor security (different vendor for each type of security attacks based on their impact and popularity) and the proposed security.

The x-axis in the given graph represents the occurrence of security attacks for the given sample requests, where 1 represents 100% and 0 represents 0%. The interpretation

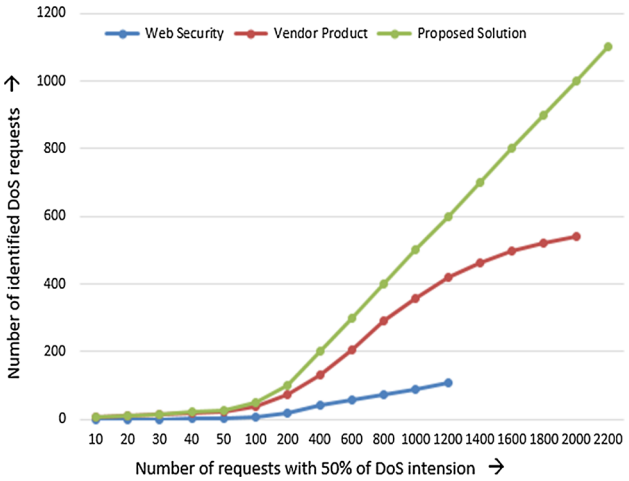

Fig. 10 Prevention of DoS/DDoS attacks on REST web services

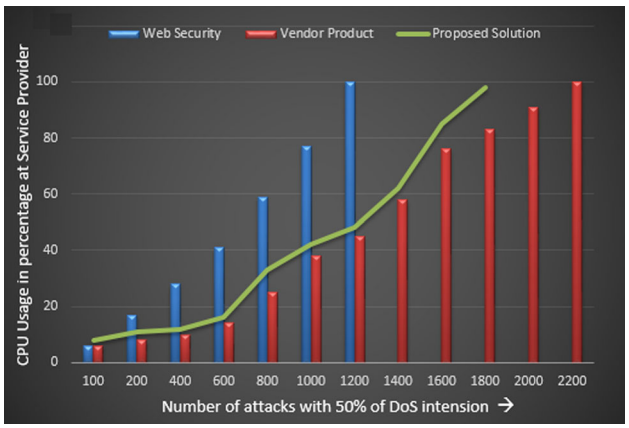

Fig. 11 CPU usage analysis at server provider

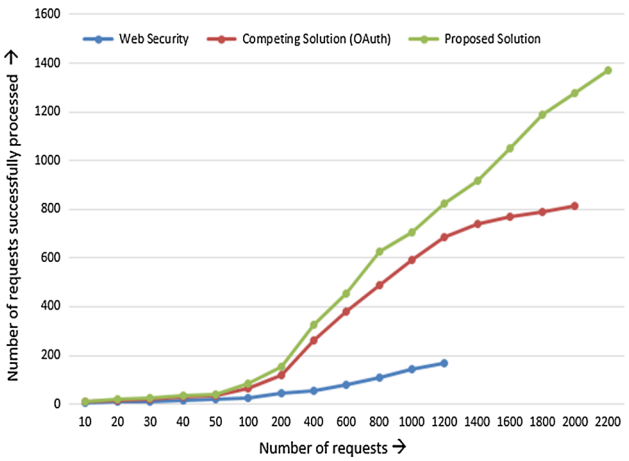

Fig. 12 REST message-level protection

of results clearly shows that the proposed security solution outperforms the OSI network layer security and vendor security. Moreover, the security seeking organizations are not required to go choosing different vendors for protecting

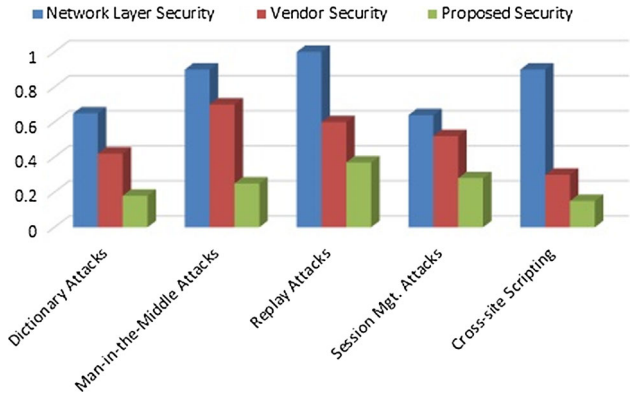

Fig. 13 Proposed security solution outperforms over network layer security and vendor security

EAI from these kind of different attacks; instead they can choose our proposed security solution that is compressive and software-based which reduces unnecessary need of additional hardware for SOA protection as some vendor products require.

6 Conclusion

As RESTful web services are not using any metadata descriptions unlike SOAP based web services, the option for applying security through metadata elements is out. Currently, the security of RESTful services relies on the network/transport layer security and ad hoc security mechanisms, which introduce additional security vulnerabilities, and reduces flexibilities. We studied the security vulnerabilities of REST principle in respect of the industry-recognized OWASP vulnerability ratings on web applications, analyzed its security requirements in the context of current generation web technologies, conducted gap analysis on existing security solutions and vendor products, and observed that none of the existing security solutions results in comprehensive security for protecting RESTful web service against SOA attacks.

In this paper, an adaptive security architecture is proposed that works on three functional principles: (1) predict, (2) prevent and (3) learn in an intelligent approach to detect any future threats, yet to respect REST architectural design. The core of the proposed security solution is constructed based on PKI and its related cryptographic technologies to secure HTTP transactions, which is the backbone of REST operations. A prototype of the proposed security algorithm was implemented using Java technologies and tested in a real-time SOA environment with large-scale sensitive financial data. The interpreted results show that the proposed security solution is very well suited for protecting REST when compared to the supported network/transport layer security, and better than the alternate solution OAuth. In addition, we found that our security solution on REST achieves higher security when compared with existing security solutions for SOAP-based web services. As our proposed solution manipulates only HTTP headers for introducing security-related data and not on the message payload unlike other security solutions do, the processing time taken for this newly introduced security-related stuff is covered mostly within the SLA of the services. As a theoretical impact, the "predict" part of the ISE uses questionnaire-based approach for the construction of security model. The practical impact that the proposed algorithm will add on SOA is the additional time consumed for the execution of security algorithms at both client and server side; however, it is within the milliseconds range at most of the time as observed by the conducted PoC results. Moreover, this latency is acceptable for security-centric enterprise environments. The conducted research can be further continued to benchmark how REST principle with this proposed security can be best utilized as a replacement to industry standard SOAP-based web services.

References

1. Sheng Z, Xiaoqiang Q, Athanasios V, Claudia S, Scott B, Xiaofei X (2014) Web services composition: a decade's overview. Inf Sci 280:218–238
2. AlShahwan F, Maha F, Godwin A (2016) Security framework for RESTful mobile cloud computing web services. J Ambient Intell Humaniz Comput 7:649–659
3. Sepulveda C, Rosa A, Jesus B (2015) QoS aware descriptions for RESTful service composition: security domain. World Wide Web 18(4):767–794
4. Fielding R (2000) Architectural styles and the design of network-based software architectures. Ph.D. Dissertation, University of California, Irvine
5. Xu B, Tianbo L, Xiaoqin W, Lingling Z, Xiaoyan Z, Wanjiang H (2013) A synthetic solution scheme for SOA security assurance. In: Proceedings of the international conference on security and management (SAM), computer engineering and applied computing (WorldComp)
6. Liu L, Wang D, Zhao J, Huang M (2013) SA4WSs: a security architecture for web services. In: Mustofa K, Neuhold EJ, Tjoa AM, Weippl E, You I (eds) Information and communication technology. Springer, Berlin, pp 306–311
7. Masood A (2013) Cyber security for service oriented architectures in a Web 2.0 world: an overview of SOA vulnerabilities in financial services. In: IEEE international conference on technologies for homeland security (HST), pp 1–6
8. Jacqui C, Marijke C (2010) Towards an information security framework for service-oriented architecture. In: IEEE information security conference. South Africa, pp 1–8
9. Kou H (2010) A study on the security mechanism for web services. In: Proceedings of the world congress on engineering and computer science, vol I, USA
10. Baghdadi Youcef (2013) A comparison framework for service-oriented software engineering approaches: issues and solutions. Int J Web Inf Syst 9(4):279–316
11. OWASP (2013) Top 10 web application vulnerabilities. Report on the ten most critical web application security risks

12. Wang Shengwei, Zhengyuan Xu, Cao Jiannong, Zhang Jianping (2007) A middleware for web service-enabled integration and interoperation of intelligent building systems. Autom Constr 16(1):112–121
13. Kim SK, Han S-Y (2006) Performance comparison of DCOM, CORBA and web service. In: Parallel and distributed processing techniques and applications conference, pp 106–112
14. Henning Michi (2006) The rise and fall of CORBA. ACM Queue 4(5):28–34
15. Jones D (2015) Cost of cyber crime study: United States. Hewlett Packard Enterprise. http://www.ponemon.org/blog/2015-cost-of-cyber-crime-united-states. Accessed 1 Mar 2017
16. Symantec (2016) Internet security threat report, vol 21. https://www.symantec.com/security-center/threat-report Accessed 1 Mar 2017
17. WhiteHat Security (2016) Web applications security statistics report. https://www.whitehatsec.com/info/website-stats-report-2016-wp. Accessed 1 Mar 2017
18. National Vulnerability Database (2016) Vulnerability metrics, National Institute of Standards and Technology, USA. https://nvd.nist.gov. Accessed 1 Mar 2017
19. McAfee Labs (2016) Threats report. https://www.mcafee.com/au/resources/reports/rp-quarterly-threats-dec-2016.pdf. Accessed 1 Mar 2017
20. Hardt D (2012) The OAuth 2.0 authorization framework. RFC 6749 (Proposed Standard), http://tools.ietf.org/html/rfc6749. Accessed 1 Mar 2017
21. Recordon D, Drummond R (2006) OpenID 2.0: a platform for user-centric identity management. In: Proceedings of the second ACM workshop on digital identity management. ACM, pp 11–16
22. Russell M (2014) Secure RESTful interface profile security analysis and guidance. The MITRE Corporation, Bedford
23. Mladenov V, Christian M, Jorg S (2015) On the security of modern Single Sign-On Protocols: second-order vulnerabilities in OpenID connect. arXiv:1508.04324
24. Ibrahim B, Fadzil MH (2016) Construction of customizable SOA security framework using artificial neural networks. J Teknol 78(12–3):69–75
25. Gartner (2016) Top 10 strategic technology trends for 2016. http://www.itbusinessedge.com/slideshows/top-10-strategic-technology-trends-for-2016-08.html. Accessed 1 Mar 2017
26. Neha L, Jwalant B (2014) DDoS prevention on REST based web services. Int J Comput Sci Inf Technol 5(6):7314–7317
27. Lee H, Mayur R (2014) Defense against REST-based web service attacks for enterprise systems. Commun IIMA 13:57–68
28. Sungchul L, Ju-Yeon J, Yoohwan K (2015) Method for secure RESTful web service. In: 14th IEEE international conference on computer and information science (ICIS)
29. Orellana F, Marko N (2012) Distributed computing with RESTful web services. In: Seventh international conference on P2P, parallel, grid, cloud and internet computing (3PGCIC), pp 103–110
30. Serme G, Anderson S, Julien M, Yves R (2012) Enabling message security for RESTful services. In: IEEE 19th international conference on web services (ICWS), pp 114–121
31. Sudhakar A (2011) Techniques for securing REST. CA Technology Exchange, New York, p 32
32. Malisetti R (2011) Securing RESTful services with token-based authentication. CA Technology Exchange, New York, pp 43–48
33. Adamczyk P, Patrick S, Ralph J, Munawar H (2011) REST and web services: in theory and in practice. In: Wilde E, Pautasso C (eds) REST: from research to practice. Springer, New York, pp 35–57
34. Brachmann E, Gero D, Klaus S (2012) Simplified authentication and authorization for RESTful services in trusted environments. In: European conference on service-oriented and cloud computing. Springer, Berlin, pp 244–258
35. Pan G, Yongbin W (2012) Securing RESTful WCF services with XAuth and service authorization manager—a practical way for user authorization and server protection. In: Fifth IEEE international joint conference on computational sciences and optimization (CSO), pp 651–653
36. Pai S, Yash S, Sunil K, Radhika P, Sanjay S (2011) Formal verification of OAuth 2.0 using alloy framework. In: IEEE international conference on communication systems and network technologies (CSNT), pp 655–659