RUHR-UNIVERSITÄT BOCHUM

Amazon Web Services Security

Analysis of S3 and SQS Interfaces

Thorsten Schreiber
Advisor: Juraj Somorovsky

Seminarthesis
March 4, 2011
Chair for Network and Data Security: Prof. Dr.-Ing. Jörg Schwenk

Abstract

This Paper is an analysis of the security measures of Amazon Web Services for the interfaces of its popular services SQS and s3.

The analysis showed that REST and Query interfaces are more secured than SOAP interfaces. For SOAP interfaces, Amazon Web Services uses a WS-Security standard that is not well implemented and a non-standard self-developed security method with conceptual flaws.

Contents

1. Introduction **1**

2. Foundations **2**
2.1. Data Formats and transition methods 2
2.1.1. XML 2
2.1.2. SOAP 2
2.1.3. REST 3
2.1.4. Query 4
2.2. Security Methods 4
2.2.1. Digital Certificates 4
2.2.2. HMAC 5
2.2.3. SSL/TLS 5
2.2.4. WS-Security 6
2.2.5. WS-Addressing 6

3. Description of Amazon Web Services **7**
3.1. Amazon Web Services Authentication 7
3.2. AWS Management Console 8
3.3. AWS Account Portal 8
3.4. S3: Simple Storage Service 8
3.5. SQS: Simple Queue Service 9

4. Evaluation of AWS Interface Security **10**
4.1. Used Software 10
4.1.1. SOAP UI 11
4.1.2. PHP Scripts and Firefox 11
4.1.3. Putty and Windows Notepad 11
4.1.4. Openssl and IBM Keystore Manager 12
4.2. S3 with REST 12
4.3. S3 with SOAP 13
4.4. SQS with Query 14
4.5. SQS with SOAP without WS-Security 15
4.6. SQS with SOAP with WS-Security 17
4.7. WS-Adressing 18

5. Conclusion **19**

A. Appendix **21**

B. Bibliography **23**

1. Introduction

A Gartner survey stated in late 2010, that "Companies spend around 10.2 percent of their budget for external IT services on cloud computing". That means, that cloud computing is an increasing factor in IT-Business. With a new interesting shared computing structure, new applications and purposes are possible.

Amazon offers many ways to use cloud computing, not only limited to run virtual instances of servers. The cloud is more than that. Amazon offers a vary range of cutting-edge platforms for web services, to bring the cloud to a new level. Away from virtual instances and shared computing power new data structures based on the cloud are created which are accessed by web applications.

A web service interface must decide, if a request from a web application is authenticated and thus valid or not. Cryptographic concepts make this decision possible. This work analyses the security of Amazon Web Service interfaces. We analyse if there are vulnerabilities in concept and implementation that allow to exploit authentication.

After discussing some foundations and presenting a description of Amazon Web services, the AWS Interface Security is evaluated. The goal of this seminar work is to evaluate the Amazon authentication mechanisms. Denial of Service and other attacks are out of scope of this evaluation.

2. Foundations

This chapter addresses the essential techniques for providing and securing access to Amazon's web services. It is divided in two sections: The first section Data Formats and Transition Methods focuses on specifications for formatting request messages and techniques for their submission. The second section Security Methods provides basic elements of cryptography that afford secure access.

2.1. Data Formats and transition methods

2.1.1. XML

The E**x**tensible **M**arkup **L**anguage [29] is an open standard for encoding documents in both machine- and human-readable form. XML is a textual data format that is divided in tags and content. The content is present as string data and is marked up by the tags. The tags assemble the logical structure of the document and are represented by a tree structure. A tag can contain meta information as attributes, e.g. an XML namespace URI.

2.1.2. SOAP

SOAP is a format standard for exchange of information in a decentralized, distributed environment. [28] In case of communication with web services, SOAP

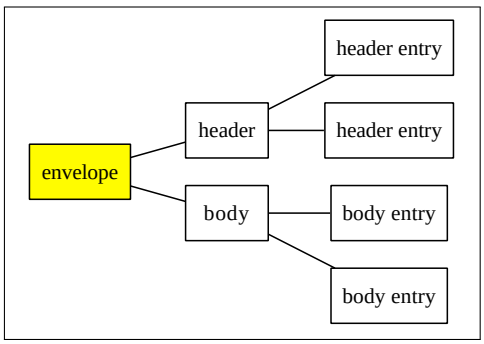

Figure 2.1.: soap structure

realizes so called remote procedure calls [25] , which allow a computer program to invoke local procedure executed on a remote computer and their responses. The basic format standard describes the structure of a framework for information exchange but does not implement any security. For data transport, SOAP protocol relies on an HTTP [20] or HTTPS [22] connection from the HTTP standard .

A SOAP message is based on an XML structure and consist of three parts. Figure 2.1 highlights the structure. A mandatory envelope as top element of the XML formatted document represents the message. The envelope contains an optional header element for extending a message to carry meta information, such as authentication or transaction management. The envelope must contain a body element, which carries information to marshal RPC calls and error reporting. A Remote Procedure Call is realized as body entry containing further entries for arguments.

The latest SOAP version is 1.2, but Amazon Web Service Interfaces use still version 1.1, like specified in the Amazon WSDL [27] file. [1]

2.1.3. REST

Like SOAP, the **Re**presentational **S**tate **T**ransfer [9] describes a method to access a web service. REST Interface has become widely accepted as a simple alternative to SOAP. IBM Software Engineer Alex Rodriquez reported in 2008 that REST "has mostly displaced SOAP and WSDL-based interface design because it's a considerably simpler style to use." [26]

Technically REST relies on HTTP framework and it is like SOAP a stateless interface for communication with web services. To marshal action or procedures, it uses the set of operations supported by HTTP. This set contains POST, GET, PUT and DELETE and its members are called HTTP verbs. Each of it stands for an action. POST,PUT,DELETE are methods to change data, GET is defined as method to query data. Note, that GET operation is defined as a safe method. That means a GET request does not allow any side effects on the web service.

A REST message pair consists of a request to a web service and its response. Every REST message follows a format scheme of header entries and body. It starts with the HTTP request line which contains the HTTP version and the HTTP verb determining the method and a resource identifier in URI form. The following header contains all information which is necessary to marshal the method call. This information is organized in fields. Every field has a name and a content. Data, e.g. files, are carried in the body of POST or PUT requests. GET and DELETE requests do not need no body.

[1] http://queue.amazonaws.com/doc/2009-02-01/QueueService.wsdl

2.1.4. Query

The Query interface is actually a REST interface, using only the HTTP-verb GET for all method calls. However, REST interface defines the GET operation as safety, that means it is not allowed to have any side effects. As modifying data will doubtless have side effects, the REST standard interdicts data modifying via GET verb. The Query interface does not seem to be a specific standard. Probably Amazon renamed the interface in this case to not infringe the REST standard. The reason to proceed all requests solely via GET may be that a web application can use the web service via Query by simply calling a URL in a web browser. Therewith the user can change data also by using the GET requests.

2.2. Security Methods

2.2.1. Digital Certificates

A public key certificate is a record, which associates a public key to an identity. To ensure integrity and authenticity and to prevent man in the middle attacks, it is important to verify the identity of a key holder. If a client wants to communicate with a server, the client will need to know if he can trust the servers public key. This identity confirmation requires a public key infrastructure as trusted third party. The trusted third party (Central Authority) is the beginning of a chain of trust, whose root certificate must be in possession of the client. The holder of the server must proof his identity against the central authority and receives in return a digital signature over his public key, which has been signed by the central authority. Based on asymmetric cryptography, the subject posses the private key.

In the context of this work, both applications of a certificate digital signature and encryption are used. For communication over a secured channel, so called transport layer security, a certificate is used to ensure that the identity of an endpoint, to which a request is posted, is verified. [23]

In order to achieve message layer security, the certificate ensures a digital signature to authenticate a request. For Amazon's implementation of SOAP WS-Security, a certificate in x.509 [23] format is required. This is an ITU-T standard for a public key infrastructure. The structure of an x509.v3 [23] digital certificate includes information about issuer, validity, subject and most important the subject's public key. The Amazon Web Service authentication logic needs this information to validate assignment of certificate to an AWS developer account and to gain the public key to verify the signature. In order to ensure message integrity and authenticity HMACs are used.

2.2.2. HMAC

A **H**ashed **M**essage **A**uthentication **C**ode [19] is a mechanism for message authentication using cryptographic hash functions. Commonly used hash functions are SHA-1 and SHA-2. Figure 2.2 demonstrates the structure of an HMAC. The function HMAC takes a private key string and a message string as input. It outputs a message authentication code digest. To prevent attachments to an existing digest, the hash function is used twice. The HMAC is secure under the assumption that the used hash-functions are cryptographic hash functions. In its security methods Amazon uses SHA-1 and SHA-256 hash functions, which are considered to meet the requirements for a cryptographic hash function.

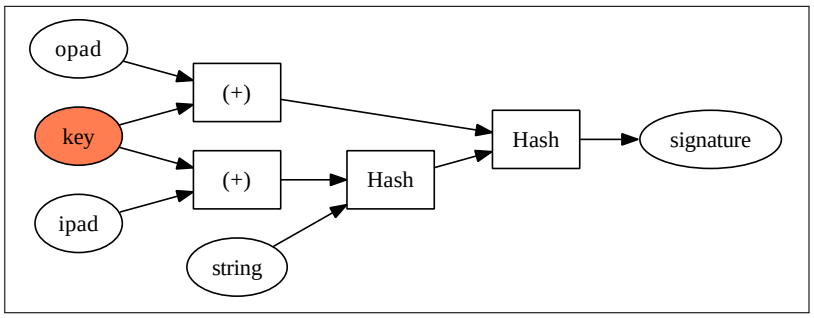

Figure 2.2.: hmac structure

To send an authenticated message between entity A and entity B, A generates an HMAC of a string with a shared key K_{AB}. Only A and B know the key. A submits the string and the HMAC digest to B. B also generates an HMAC of the submitted string with the same key. If the digests of A and B are identical, the digest is verified. Amazon will treat the digest as signature.

2.2.3. SSL/TLS

Secure **S**ockets **L**ayer is a **t**ransport **l**ayer [21] extension to provide a **s**ecure communication channel between two computer entities over the Internet. To achieve the security targets privacy, confidential, authentication and integrity, the security layer encrypts the HTTP messages. In the common setup, the client expects a certificate from the server to ensure its identity. Although SSL standard supports client certificates for authenticating the client's public key by the server, this feature is not used. The sole responsibility to verify identity of the server is at the client. If the identity is not verified well, a man in the middle attack is possible and the communication channel is compromised. The protocol itself seems to be considered as secure, although attacks exist against implementation of the protocol mostly attacking the identity verifying.

2.2.4. WS-Security

The Web Service Security [15] extends SOAP messages with security features. It belongs to the WS-* Suite of web service specification published by OASIS. To achieve confidentiality and integrity, the WS-Security standard defines a message layer security. For Amazon Web Service authentication purposes the digital signature feature and the binary security token feature are used.

The binary security token [16] is a security token in binary format. An X.509 certificate requires a special encoding format for inclusion in XML formats. The binary security token is included with `wsse:BinarySecurityToken` element. The element attribute EncodingType specifies the encoding. In the Amazon case it is Base64Binary. It contains a BASE64 encoded X.509 certificate, which Amazon's verification logic takes to identify the public key.

The digital signature feature of WS-Security is a specification how to integrate signature of relevant XML elements of SOAP Message. It is located in a `ds:Signature` tag. The elements which shall be signed are identified by a `wsu:Id` attribute. For each signed element in SOAP message there is a ds:reference element which carries information of canonicalization algorithm, hash algorithm and a `wsu:Id` identifier. The reference elements are child nodes in a `ds:signedInfo` tag, which is finally signed by the private key of requester. The resulting signature is stored in `ds:SignatureValue` element.

Both features are encapsulated by the `wsse:Security` element and are called header block. The `wsse:Security` element is present in the `soap:Header`, because the header is defined as the location for meta information.

Amazon's SQS interface logic uses WS-Security with SOAP to validate the authenticity of the incoming requests.

2.2.5. WS-Addressing

WS-Addressing [14] feature is also a part of the WS-* Suite. It ensures that a SOAP Message reaches it intended endpoint by including the endpoint in the soap:Header. The minimal set of information required by Amazon are the elements `wsa:To`, `wsa:MessageID`, `wsa:ReplyTo` and `wsa:Action`. The content of `wsa:To` element is the address of the intended receiver of this message. The content of `wsa:MessageID` element is a URI that uniquely identifies a message in time and space. The content of `wsa:ReplyTo` element is an endpoint reference that identifies the intended receiver for responses to a message. The content of `wsa:Action` is an identifier that uniquely and opaquely identifies the requested function calls implied in the a message body.

All elements are immediate childes of the `soap:Header` element node.

3. Description of Amazon Web Services

Amazons web services [1] were launched in July 2002 [7] and meanwhile provide a wide collection of web services. Amazon offers the web services to developers for building their own web applications which are based on Amazon Web Services.

All services are cloud based and rely on Amazon's global computing infrastructure. Amazon offers popular web services in categories such as computation, messaging, databases and networking. Very popular web services are EC2 and S3, which offer together a platform for running virtual machines in the Amazon cloud.

3.1. Amazon Web Services Authentication

To ensure that only a trusted person or entity posts request, Amazon's web services require authentication. A positive authentication derives from the fact that Amazon's web service trusts the requesting entity to be in possession of a certain secret.

Amazon uses the digital signature technique for message authentication. Two different methods to create a signature are used. For both methods an AWS Developer Account is needed.

The non standard authentication mechanism is developed by Amazon and is a signature of a special string. This string is called canonic string and is discussed later. The signature mechanism is based on symmetric cryptography, hence there is a secret key shared between Amazon and the requesting entity. The secret key and its identifier are associated with an AWS Developer Account. The secret key is an alphanumeric string of 40 characters, the public key identifier is a 20 character string. It is specified by AWS that an HMAC cryptographic function uses the secret key to sign the canonic string and outputs a signature. According to Amazon's standard the HMAC function uses SHA1 or SHA256 hash functions.

The described non standard method is used for the interfaces as follows: S3 and SQS SOAP interface use the method with SHA1 and SQS Query interface with SHA256.[2]

The second authentication mechanism follows the standard and uses SOAP Messages WS-Security standard. From this WS-Security suite, Amazon takes the signature scheme and the binary security token pattern. The signature scheme of WS-Security is based on asymmetric cryptography. Amazon realizes it with an X.509 certificate, which can be created by the developer or by Amazon. The requesting entity keeps the private key belonging to the certificate and shares the public key and certificate with Amazon. The certificate is assigned once with the AWS developer account and must be submitted as binary security token in each request. This security standard is used for the SOAP SQS interface.

As Amazon reviews continuously its algorithm to security, different versions may exist. This analysis focuses on the current algorithms.

3.2. AWS Management Console

The Amazon Management console [6] is a web based user interface and offers a direct and simple way to access the most of Amazon's web services, without installing or implementing software. To access it and Amazon's web services, a developer must own an amazon.com account and extend it to a developer account. For authentication, the Amazon login and password are required. A secure connection is ensured by HTTPS [22].

3.3. AWS Account Portal

The AWS Account Portal [5] is the central management web GUI for account configuration. It provides features like setting payment method, viewing consolidated billing and usage reports. Relevant to security is the management of the security credentials granting access the web services. The platform generates symmetric keys and offers a revocation mechanism. In the same way, the Amazon Account Portal generates certificates or receive them and assigns or deassigns them to the AWS account. Like AWS Management Console, the AWS Account Portal is secured by login and password and uses an HTTPS connection.

3.4. S3: Simple Storage Service

Amazon S3 **S**imple **S**torage **S**ervice [3] is a web service, which stores files in the Amazon cloud. The costumers pay only for use, the amount of used storage and the caused traffic. The files are stored in so called buckets. Each bucket is identified by a unique name. The bucket can contain files and folders. Amazon

lets the owner set file access permissions in order to allow a granulated managed file sharing. This offers a wide range of applications.

This web service accepts request via SOAP and REST over HTTP and HTTPS.

3.5. SQS: Simple Queue Service

Amazon SQS **S**imple **Q**ueue **S**ervice [4] is a system to exchange messages between web services asynchronously. It is a web service based on a queue and may be viewed as a cloud data structure. Each queue holds all send messages until they are received and deleted. According to Amazon's pay per use principle, only the amount of messages is charged.

This web service allows to post request via Query Interface and SOAP Interface, The SOAP interface offers two security methods. All SQS SOAP interfaces must use secure HTTPS connection.

4. Evaluation of AWS Interface Security

In this chapter, we describe the used software and our methods. Finally we present the results of the evaluation. Table 4.1 shows the tested interfaces of the web services and the results of the evaluation and gives an overview about found security issues. We present the results in this table as a preview, the detailed proceedings and their results are given in Sections 4.2 to 4.7.

The web service SQS does not support REST interface and the web service S3 does not support Query or SOAP with WS-Security standard. The entry `(HTTPS)` shows that the web service is secured by a transport layer. The entry `(HTTP)` shows, that the web service is solely secured by message layer. If both entries `(HTTP,HTTPS)` are listed, the developer is free to decide between submitting a request via unsecured HTTP or via secured HTTPS.

Interface \Webservice	SQS	S3
REST	✗	no issues found [4.2] (`HTTP, HTTPS`)
Query	no issues found [4.4] (`HTTP, HTTPS`)	✗
SOAP non-standard	unauthorized arguments [4.5] (`HTTPS`)	unauthorized arguments [4.3] (`HTTP, HTTPS`)
SOAP WS-*	Wrapped Body Vulnerability [4.6] WS-Adressing not working [4.7] (`HTTPS`)	✗

Table 4.1.: Available Interfaces and Results

4.1. Used Software

This section lists and explains the different tools used during the tests.

4.1.1. SOAP UI

SOAP UI is a tool to test web services. [8] Its main purpose is to compose SOAP requests and display the responses to those requests. In this work, it was used to generate and send the SOAP requests to the Amazon Web Service endpoints and display the response. SOAP UI uses WS-Security features automatically, based on stored security information in a Java Key Store [17]. It is a certain file format from Java language to store security information. In SOAP UI, the XML structure of SOAP requests can be manipulated comfortably in raw format before sending.

4.1.2. PHP Scripts and Firefox

Unfortunately SOAP UI does not support Amazon's non standard SOAP authentication mechanism, it must be calculated manually. For this purpose, a small Linux command line script in PHP [10] was used. The script takes a canonic string as input, which must be manually concatenated, and outputs the corresponding signature as base64 encoded data to the standard output. It is copied and pasted to the appropriate XML tag in SOAP UI SOAP request window.

Due to the fact, that it is only a testing script and runs in secured perimeters, the key is hard coded. The script uses the cryptographic function HMAC of PHP library to generate the signature. The script is presented in Listing 4.1

```
1 $canonicString = $_SERVER['argv'][0];
2 $key = "....";
3 return base64_encode( hash_hmac('sha1',$canonicString, $key,true) );
```

Listing 4.1: Calculating AWS non standard SOAP Signature

A second script was used to test the Amazon Query interface. It must be installed on a web server and offers a web-GUI. It displays two text areas, one with all variables which should be signed and another one with additional arguments an adversary would append to compromise the query request. The request is represented as a hyper link and can be send by opening a link with Firefox web browser. The source code of the script is given in the Appendix in Listing A.1.

4.1.3. Putty and Windows Notepad

The software putty is a terminal application for SSH (SecureSHell, a remote terminal for Linux systems) connections. But it is also able to open a raw HTTP connection to a REST interface and to send the content of the clipboard to an endpoint. Due to some format reasons with SOAP UI it was easier and faster to build and store the request with Microsoft's text editor notepad and send those using putty.

4.1.4. Openssl and IBM Keystore Manager

Amazon generates certificates for developers in pem format, which is a base64 encoded version of a PKCS7 [24]. To store it to a Java Key Store, openssl must convert pem certificate format to PKCS12 [12] format on Linux command line in Listing 4.2. Amazon Security Console offers only to download the private key and the certificate in two separate files, where cert.pem contains the certificate.

```
1 openssl pkcs12 -export -in cert.pem -inkey key.pem -name "aws" -out
    cert.p12
```

Listing 4.2: Converting pem to PKCS12

The native Java Key Store manager does not support the import of private keys. The IBM Key Manager [11] does. Guided by a graphical user interface, the import can be done quickly.

4.2. S3 with REST

Listing 4.3 presents a sample REST request to upload an image via HTTP's PUT method. For authentication Amazon requires an HMAC signature with the AWS private key of a canonic string's SHA-1 digest. The AWS key identifier and the signature are submitted in HTTP's Authorization header. The canonic string contains the HTTPP verb, a timestamp, the full file name, the content type and the bucket name. Hence, the request data are authenticated, but not the complete content of the uploaded file.

It was observed, that signature of hash digest of the upload file is not mandatory. If a developer forgets about to create and include the MD5 digest of the payload to the signature, an adversary in possession of the original request can manipulate the upload file within lifetime of the timestamp. Even if he is limited to signed file type header field, he can overwrite the original file with a file of similar type.

```
1 PUT /my-image.jpg HTTP/1.1
2 Host: myBucket.s3.amazonaws.com
3 Date: Wed, 12 Oct 2009 17:50:00 GMT
4 Authorization: AWS ID:SIGNATURE=
5 Content-Type: image/jpeg
6 Content-Length: 11434
7 Content-Md5: sRLHbQKy6pQ==
8
9 [The binary encoded Image]
```

Listing 4.3: Sample REST Request

Regarding implementation, no attacks compromising authentication could be found. Attempts with duplicated header entries, oversized signatures, left-out signature and incorrect date format did not succeed.

4.3. S3 with SOAP

The S3 SOAP interface accepts connections over HTTP and HTTPS. Listing 4.4 presents a sample SOAP request. For authentication, WS-Security methods provided by the standard are not used. Moreover, the security information is not carried in the header according to SOAP standard. The security information is carried in body. So the header is not used and therefore optional. The security information is passed as the remote procedure call's arguments. It consists of Timestamp, AccessKey-Id and Signature. Each is a child element of the method invocation node, see row 6 to 8. Amazon uses a non-standard security method based on an HMAC with a SHA-2 hash algorithm and a 40-char private key to sign a canonic string. This canonic string is a simple concatenation of the method name and a timestamp. As follows, only the timestamp and the method name are signed. The timestamp ensures message freshness and after expiring also prevents replay attacks. During its lifetime replay attacks are possible because Amazon does not safeguard against repeated reception of the same request. The signing of the method name binds the timestamp to the method and ensures no other method is called.

This concept should ensure authenticity but turned out to have flaws because the arguments are not signed and therefore unauthenticated. Using this flaw, an adversary can capture a signed message and manipulate the arguments and make a valid request within timestamp lifetime and on the same method.

```
1  <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/
       envelope/" xmlns:ns="http://s3.amazonaws.com/doc/2006-03-01/">
2   <soapenv:Header/>
3   <soapenv:Body>
4      <ns:CreateBucket>
5         <ns:Bucket>UnsignedName</ns:Bucket>
6         <ns:AWSAccessKeyId>XXXXXXXXXXXXXXXXXXXX</ns:AWSAccessKeyId>
7         <ns:Signature>ek5qM8Q05LbR48CMFKvzXo+qtXA=</ns:Signature>
8         <ns:Timestamp>2011-01-06T12:06:19Z</ns:Timestamp>
9      </ns:CreateBucket>
10  </soapenv:Body>
11 </soapenv:Envelope>
```

Listing 4.4: Sample S3 SOAP Request

An adversary can manipulate the request of listing 4.4 and is able to create new buckets within the timestamp's lifetime which is half an hour. According to error messages, Amazon validates timestamps with S3 SOAP interface which are up to 15 minutes old and also timestamps which can be up to 15 minutes in the future. Hence, the complete lifetime lasts 30 minutes.

It is a serious problem that Amazon does not require transport layer security and accepts non-SSL connections. That makes it easy for an adversary to cache a request, manipulate and replay it.

The application logic expects the security information to be on the specified place in SOAP structure. Other placements are ignored. The implementation was also tested by doubling the security arguments. If a request contains multiple timestamps, the interface logic regards the oldest. If the signature or AWS-Key ID is doubled, the interface takes the last occurrence of a security information.

Moreover, tests showed the sequence of validating a request. First, the interface logic validates the XML-scheme. The request must contain exactly one body-tag with one child-Element. A header is optional and not regarded. Secondly, the timestamp is evaluated. Thirdly, the authentication logic calculates and validates the signature. On positive validation the requested method is triggered.

4.4. SQS with Query

Amazon uses a similar procedure to sign Query request. Amazon had to review its SQS Query Interface signature algorithm due to security reasons. [18] Thus, there are more versions of the signature algorithm. The current version is 2. An HMAC using the SHA-256 hash function and the symmetric private key signs a canonic string. This canonic string is a concatenation according to an algorithm from Amazon. Ultimately, the canonic string must contain each submitted value. Hence, the complete request is signed and so authenticated.

```
1 http://queue.amazonaws.com/?AWSAccessKeyId=XXXXXXXXXXXXXXXXXXXX&
    Action=ListQueues&SignatureMethod=HmacSHA256&SignatureVersion=2&
    Timestamp=100000000-10-22T12%3A00%3A00.000Z&Version=2009-02-01&
    Signature=XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX%2BN2ambqcg%3D
```

Listing 4.5: Sample SQS Query Request

```
1 GET\n
2 queue.amazonaws.com\n
3 /\n
4 AWSAccessKeyId=XXXXXXXXXXXXXXXXXXXX
5 &Action=ListQueues
```

```
6 &SignatureMethod=HmacSHA256
7 &SignatureVersion=2
8 &Timestamp=2010-10-22T12%3A00%3A00.000Z
9 &Version=2009-02-01
```

Listing 4.6: canonic string

Regarding to the implementation, it is not possible to add unsigned elements to overwrite signed informations after validation.

The concept and tested implementation meet security targets integrity and authenticity because the whole GET-REQUEST is signed and can not be manipulated under assumption that the HMAC is not broken.

Another probem is the unlimited lifetime of a timestamp. The application logic accepts the timestamp, if it is in the future, no matter how long the date is away. The developer can create a timestamp with a future date and sign it, and the application logic will accept it.

4.5. SQS with SOAP without WS-Security

The SQS SOAP interface uses the same security method as S3 with SOAP (Section 4.3). However, different error messages indicate a different implementation of interface logic. Further, the canonic string is build differently. It is a concatenation of the method name and the timestamp. The service name is not included, unlike S3 with SOAP.

The interface logic proceeds another sequence of validating. First, the SOAP scheme is validated. Secondly, the signature is calculated and verified. Thirdly, the timestamp is validated. On success, the method included in the SOAP body is triggered.

```
1  <soapenv:Envelope
2        xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
3        xmlns:ns="http://queue.amazonaws.com/doc/2009-02-01/">
4   <soapenv:Header
5     xmlns:aws="http://security.amazonaws.com/doc/2007-01-01/">
6    <aws:AWSAccessKeyId>XXXXXXXXXXXXXXXXXXXX</aws:AWSAccessKeyId>
7    <aws:Timestamp>2010-11-29T12:20:00Z</aws:Timestamp>
8    <aws:Signature>Y2KVm4k8dTmiFKfGUHvvSv0+K74=</aws:Signature>
9   </soapenv:Header>
10 <soapenv:Body>
11     <ns:CreateQueue>
12          <ns:QueueName>MyQueNEXT</ns:QueueName>
13          <ns:DefaultVisibilityTimeout>40</ns:DefaultVisibilityTimeout
                >
14       </ns:CreateQueue>
15  </soapenv:Body>
```

```
16 </soapenv:Envelope>
```

Listing 4.7: Sample SQS SOAP Request without WS-Sec

Like the S3 SOAP interface the arguments are not authenticated, as the security policy dictates only signature over timestamp and method name. Hence, the same attack vector from S3 with SOAP interface is possible with one limitation: A mandatory transport layer security. Testing access per HTTP affected an error message that HTTP is deprecated. The submission of a request must be secured by a SSL connection.

Another fact is, that a developer can extend the lifetime of an timestamp to nearly infinity, due to application logic validating a timestamp containing a future date unlimited as true. The developer can create a timestamp with a date like 11/2/10000000000, 13:35:00 o'clock, sign it, and the application logic will accept it. Refer Listing 4.8 and 4.9 for request and response.

```
1  <soapenv:Envelope xmlns:ns="http://queue.amazonaws.com/doc
       /2009-02-01/" xmlns:soapenv="http://schemas.xmlsoap.org/soap/
       envelope/">
2    <soapenv:Header xmlns:aws="http://security.amazonaws.com/doc
         /2007-01-01/">
3      <aws:AWSAccessKeyId>XXXXXXXXXXXXXXXXXXXX</aws:AWSAccessKeyId>
4      <aws:Timestamp>10000000000-02-11T13:35:00Z</aws:Timestamp>
5      <aws:Signature>L+l7Ye9UgBaACdjIElJEk6Kj0PE=</aws:Signature>
6      </soapenv:Header>
7      <soapenv:Body >
8        <ns:ListQueues></ns:ListQueues>
9      </soapenv:Body>
10   </soapenv:Envelope>
```

Listing 4.8: SQS's Never Ending Timestamp Request

```
1  <soapenv:Envelope xmlns:soapenv="..">
2   <soapenv:Body>
3      <ns:ListQueuesResponse xmlns:ns="..">
4         <ns:ListQueuesResult>
5            <ns:QueueUrl>https:// ... </ns:QueueUrl>
6        ...
7            <ns:QueueUrl>https:// ... </ns:QueueUrl>
8         </ns:ListQueuesResult>
9         <ns:ResponseMetadata>
10           <ns:RequestId>XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX</ns:
                 RequestId>
11        </ns:ResponseMetadata>
12     </ns:ListQueuesResponse>
13  </soapenv:Body>
14 </soapenv:Envelope>
```

Listing 4.9: SQS's Never Ending Timestamp Answer

4.6. SQS with SOAP with WS-Security

The SQS SOAP interface allows the use of the standard extension WS-Security to secure SOAP-messages. Amazon's standard [2] takes the signature mechanism to authenticate requests. Amazon's standard requires as minimum a signed timestamp, if using an HTTPS connection. At test time, no HTTP connections were allowed anymore. Amazon's standard recommends to sign the Action header element, if WS-Adressing is used.

The interface logic was tested against wrapping attack. In this case the body and the timestamp were signed.

Recall that wrapping attacks rely on a communication flaw between validating and executive logic while handling a partly signed message. The validating logic does not communicate which parts are signed and verified. But the executive logic adopts the whole message as verified and executes the method.

To test the interface against timestamp wrapping attack, a SOAP Message with an expired timestamp but valid signature was taken. The private signing key is unknown. The signed timestamp is moved to another place in the document and

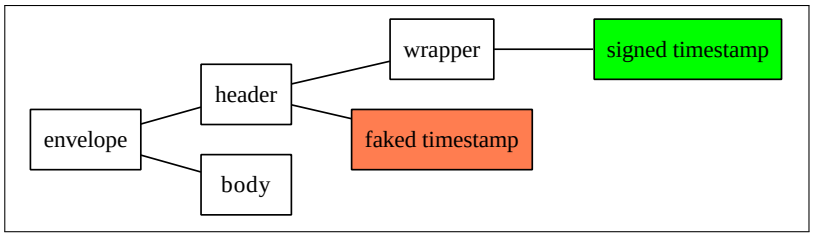

Figure 4.1.: wrapped timestamp structure

a new, valid timestamp is generated and included at the original timestamp's position. Goal is that validating logic verifies the moved timestamp and the executive logic uses the included timestamp. Amazon prevents this attack by inspecting the exact position of timestamp. A timestamp is only accepted, even for validating, if it is located as a direct child in the SOAP-Security header element. A wrapped timestamp attack is not possible to our best knowledge. Figure 4.1 highlights the positions in XML document structure.

To test a wrapped body attack, the signed SOAP-body was copied in a wrapper element in header. This one will be verified. As a next step, the `wsu:Id` property of the SOAP-body on original location was manipulated. So, the SOAP Message compiled to standard, due to a unique `wsu:id`. After that, the body including the invoked method could be manipulated and the application logic executed the request.

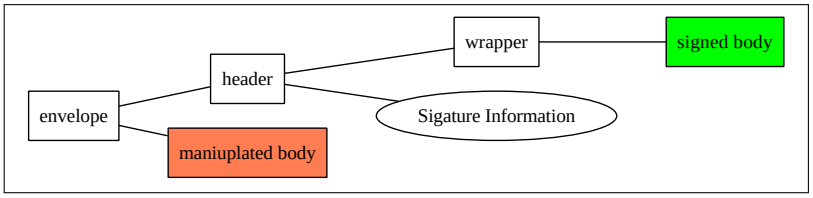

Figure 4.2.: wrapped body structure

4.7. WS-Adressing

At both non-standard and WS-* SQS with SOAP interfaces, there is a certain vulnerability concerning the redirection of messages. As a request is secured by HTTPS, the vulnerability cannot be exploited. But under the assumption, the HTTPS connection could be corrupted with a certificate validation error [13] following attack was possible.

In SOAP Messages, the message destination is not included in the message itself even with WS-Security standard. If an adversary can cache a request he can send it to another endpoint without manipulating the request SOAP Message. If mounted against SQS, an adversary can relay messages intended for a certain queue to a different queue of the victim.

The attack should be prevented by WS-Adressing Feature of WS-Security. To include the address in the signed message, Amazon standards use WS-Adressing. The tests indicated that the WS-Adressing feature is not yet usable. A WS-Adressing header requires the elements `wsa:Action`, `wsa:ReplyTo`, and `MessageID`.

The tests showed no field is validated by interface logic, except the `wsa:ReplyTo` content. It must contain the w3c anonymous address string. The other fields can contain arbitrary strings.

Even if it is listed in the Amazon Web Services documentation [2], results show that it does not work.

5. Conclusion

In this paper we investigated the security of Amazon Web Service interfaces. Our analysis showed that the security of these interfaces is not sufficient at all.

The non standard SOAP authentication method developed by Amazon has serious flaws due to unauthenticated arguments. An adversary may catch requests and replay them in lifetime of timestamp with manipulated arguments. Available to secure both web service interfaces of S3 and of SQS, only SQS interface dictates a transport layer security to mitigate the risk of request theft. The S3 interface still accepts unsecured connection and therefore the security on message layer is insufficient. This security flaw on message layer derives from a faulty concept and not from implementation errors. It is important to sign the parts of a message, which are critical and treated in a later process, here the procedure call. To produce an all-embracing message security layer, Amazon has to improve its concept and include the arguments in the set of signed elements.

The SOAP message security extension WS-Security is a standard concept. It must be implemented with paying attention to the prevention of XML wrapping attacks. Due to signatures by `wsu:Id` reference, the multiple existence of an identifier must be tested. Another failure is to regard the whole request as trusted, even if it is partly signed. The investigation showed, that the timestamp wrapping attack was prevented by Amazon. The wrapped body attack works, which shows that the implementation is not correct at all. Working wrapped body attack and fixed wrapped timestamp attack and the fact, that the minimum requirements for SOAP with WS-Security authentication are signing the timestamps suggest that Amazon does not classify the attack as security sensitive.

As discussed before, the security of the SOAP Messages is mainly derived from transport layer security, due to incorrect implementation or concept flaws. If securing those flaws with SSL is a good solution has to be discussed in future, especially regarding to cloud computing structures.

The SQS Query and the S3 REST interfaces seem to be sufficient secured. All critical request parts are signed. The security concept of SQS Query interface dictates that all information is signed, in contrast to the S3 REST concept that allows the developer to sign all critical information. Therefore the developer is responsible to choose the best security measures. Important is, that the platform allows using sufficient and working security measures. Communication with SQS and S3 interfaces provides this possibility. The results demonstrate also a

vulnerability at SQS's non standard SOAP and Query interface. A timestamp is accepted no matter how far its date is in the future. Under the presumption that an adversary could manipulate the system time of a requesting web service and cache requests through SSL certificate validation errors, he could get in possession of a request which is valid super long time. This could be prevented by specifying a maximal time difference, as it is already implemented in S3 interface.

In conclusion, the interfaces using REST and Query are better secured than SOAP interfaces. Perhaps this fact correlates to the popularity of REST. The greater the number of accesses to an interface is, the more a good working security pays off. The popularity follows from the easy-to-use design principle of REST. An example for the popularity of REST is that the popular book "Programming Amazon Web Services" by O'Reilly concentrates on REST and Query interfaces rather than SOAP interfaces.

There are less improvements to do at these popular interfaces. Regarding the security of SOAP, there is much work left to be done. To offer a highly secured web service, the implementation of WS-Security extension and the non standard authentication concept must be revised.

A. Appendix

```
<?
  $key = "c3tVLymdjz/lw9AbPB45QWCwSLkWmY7hEXu6esnK";

  function hmacsha256($key,$data)
           {
           return hash_hmac('sha256',$data, $key,true);
           }

   function hmacsha1($key,$data)
           {
           return hash_hmac('sha1',$data, $key,true);
           }

  function  b_hmacsha256($key,$data)
           {
            return base64_encode(hmacsha256($key, $data));
            }

  function  b_hmacsha1($key, $data)
           {
           return base64_encode(hmacsha1($key, $data));
           }

   function remove_cr($string)
  {
       $findit = array ("\r","\n");
       $interim = (string)str_replace($findit, "", $string);
       $interim = (string)str_replace("\\n", "\n", $interim);
       return $interim;
  }
  function makeget($getcall)
  {
      $getcall  = str_replace("\n", "", $getcall);
      $getcall  = str_replace("\r", "", $getcall);
      $getcall  = str_replace("GET", "http://", $getcall);
      $getcall  = str_replace("AWSAccessKeyId",
                    "?AWSAccessKeyId", $getcall);

      return $getcall;
    }
?>
<html>
```

```
43   <header><title >AWS Query</title></header><body>
44
45      <form method=get>
46 <textarea name="text" cols="70" rows="10" >
47 <?
48 echo stripslashes($_GET['text']);
49  if (!isset($_GET['text']))
50  {
51  echo 'GET\n
52 queue.amazonaws.com\n
53 /\n
54 AWSAccessKeyId=XXXX
55 &Action=ListQueues
56 &SignatureMethod=HmacSHA256
57 &SignatureVersion=2
58 &Timestamp=2010-10-22T12%3A00%3A00.000Z
59 &Version=2009-02-01';
60 }
61 ?></textarea>
62       <textarea name="appendix" cols="70"
63            rows="10" ><? echo stripslashes($_GET['appendix'])?></
                textarea>
64       <input type=submit>
65       <br>
66       <br>
67 <?
68
69 if (isset($_GET['text']))
70 {
71       $signstring = remove_cr(stripslashes($_GET['text']));
72       $signature1 = b_hmacsha1($key, $signstring);
73       $signature256 = b_hmacsha256($key, $signstring);
74       $getcall = makeget($signstring);
75       echo '<a href="'.$getcall."&Signature=".urlencode($signature1)
76             .$_GET['appendix']
77               .'" target="request_answ">send get with SHA1</a><br
                    ><br>';
78       echo '<a href="'.$getcall."&Signature=".urlencode(
             $signature256)
79               .$_GET['appendix']
80               .'" target="request_answ">send get with SHA256</a>'
                      ;
81
82  }
83
84       ?>
85     </form>
86   </body>
87 </html>
```

Listing A.1: Calculating AWS non standard SOAP Signature

B. Bibliography

[1] Amazon. Amazon web services, December 2010. URL http://aws.amazon.de/.

[2] Amazon. Making request authentication, December 2010. URL http://docs.amazonwebservices.com/AWSMechanicalTurkRequester/2008-08-02/index.html?MakingRequests_RequestAuthenticationArticle.html.

[3] Amazon. Simple storrage service documentation, December 2010. URL http://aws.amazon.com/s3/.

[4] Amazon. Simple queue service documentation, December 2010. URL http://aws.amazon.com/sqs/.

[5] Amazon. Amazon account portal, 2010 dec. URL https://aws-portal.amazon.com/gp/aws/developer/account/index.html.

[6] Amazon. Amazon management console, 2010 dec. URL http://aws.amazon.com/console/.

[7] GARTNER Andrews. Amazon offers free web services to help drive site traffic, July 2002. URL http://www.gartner.com/resources/108400/108427/108427.pdf.

[8] eviware. Soap ui, December 2010. URL http://www.eviware.com/soapUI/soapui-products-overview.htm.

[9] Thomas Fielding. Architectural styles and the design of network-based software architectures, November 2008. URL http://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf.

[10] The PHP Group. Php language homepage, 2010 dec. URL http://www.php.net/.

[11] IBM. Keyman, feb 2007. URL http://www.alphaworks.ibm.com/tech/keyman.

[12] RSA Laboratories. Pkcs 12 v1.0: Personal information exchange syntax, jun 1999. URL ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-12/pkcs-12v1.pdf.

[13] Moxie Marlinspike. Null prefix attacks against ssl / tls certificates, jul 2009. URL http://www.thoughtcrime.org/papers/null-prefix-attacks.pdf.

[14] OASIS. Web services addressing (ws-addressing), August 2004. URL http://www.w3.org/Submission/ws-addressing/.

[15] OASIS. Soap message security 1.1, February 2006. URL http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf.

[16] OASIS. Web services security x.509 certificate token profile 1.1, February 2006. URL http://www.oasis-open.org/committees/download.php/16785/wss-v1.1-spec-os-x509TokenProfile.pdf.

[17] Oracle. Java cryptography architecture (jca) reference guide, feb 2011. URL http://download.oracle.com/javase/6/docs/technotes/guides/security/crypto/CryptoSpec.html.

[18] Percival. Aws signature version 1 is insecure, dec 2008. URL http://www.daemonology.net/blog/2008-12-18-AWS-signature-version-1-is-insecure.html.

[19] RFC. Hmac: Keyed-hashing for message authentication, February 1997. URL http://www.ietf.org/rfc/rfc2104.txt.

[20] RFC. Hypertext transfer protocol – http/1.1, 1999. URL http://tools.ietf.org/html/rfc2616.

[21] RFC. The tls protocol, January 1999. URL http://www.ietf.org/rfc/rfc2246.txt.

[22] RFC. Hypertext transfer protocol – http/1.1, May 2000. URL http://tools.ietf.org/html/rfc2817.

[23] RFC. Internet x.509 public key infrastructure certificate and certificate revocation list (crl) profile, May 2008. URL http://www.ietf.org/rfc/rfc5280.txt.

[24] RFC. Cryptographic message syntax (cms), sep 2009. URL http://www.ietf.org/rfc/rfc5652.txt.

[25] RFC. Rpc: Remote procedure call protocol specification version 2, December 2009. URL http://tools.ietf.org/html/rfc5531.

[26] IBM Rodriquez. Restful web services: The basics, November 2008. URL https://www.ibm.com/developerworks/webservices/library/ws-restful/.

[27] W3C. Web services description language (wsdl) 1.1, May 2000. URL http://www.w3.org/TR/wsdl.html.

[28] W3C. Making request authentication, December 2010. URL http://www.w3.org/TR/2000/NOTE-SOAP-20000508/.

[29] W3C. Making request authentication, December 2010. URL http://www.w3.org/TR/xml/.