HTTP Signature Authentication Library

A thesis submitted in partial fulfillment of
the requirements for the degree of

Dual Degree Summer Internship Report

in

Computer Science and Engineering

by

Amatya Sharma
(Roll No. 17CS30042)

Under the guidance of
Server Technology, Oracle Bangalore

Computer Science and Engineering
INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR
West Bengal, India
April 2021

Certificate

ORACLE

Oracle India Pvt. Ltd.
India Development Center

Oracle Technology Park
3, Bannerghatta Park
Bangaluru - 560 029

CIN: U74899DL1993PTC051764
Phone +91 80 4107 0000
Fax +91 80 2552 6124

Private & Confidential

Ref: Oracle-Interns- 1440745

27th July 2021

TO WHOMSOEVER IT MAY CONCERN

This is to certify that Mr.Amatya Sharma , student of IIT Kharagpur (Pursuing B.Tech (Computer Science Engineering)). has completed his project with Oracle India Private Limited.

The project was undertaken from 19-May-2021 to 13-Jul-2021.He worked on "HTTP Signature Authentication Java Library"

We wish him all the best in his future endeavours.

Yours sincerely

For Oracle India Private Limited.

Rambabu Jagatha
Director – HR Operations

Acknowledgements

I would like to thank my mentor, Shankar Venugopal, for his exceptional guidance and support, without which this project would not have been possible. He has always motivated me to explore as much as I can, look into multiple implementations and sources, and try as many ideas as I can. He has always supported me through whatever problems I faced during the project.

I would also like to thank my manager, Mohanna Kera, for his directions and support. He has always kept a check over the progress of my project and worked as a constant source of motivation throughout the process.

In conclusion, I recognize that this project would not have been possible without the support from Server Technology, Oracle, Bangalore as well as the Department of Computer Science and Engineering, IIT Kharagpur. Many thanks to all those who made this project possible.

Amatya Sharma

Abstract

There have been multiple authentication schemes over the last decade to secure HTTP messages transmitted between the client and the server, including TLS, Basic HTTP Authentication, OAuth. But none of those have completely satisfied the broader problem of authenticating the sender of a particular message while keeping their identity secure, and in the process minimize the Transmission Errors in the message, and ensuring that all this is done in a simpler and cleaner implementation to ensure widespread use and standardized approach. The HTTP Signature Authentication Scheme answers all the aforementioned questions. Several web service providers have invented their own schemes for signing HTTP messages, but to date, none have been standardized. While there are no techniques in this proposal that are novel beyond the previous art, it is useful to standardize a simple and cryptographically strong mechanism for digitally signing HTTP messages. Our implementation provides a standardized implementation of the HTTP Signature Authentication Technique, specified in Oracle Drafts by M. Cavage et al.

Contents

Chapter 1

Introduction

When communicating over the Internet using the HTTP protocol, it can be desirable for a server or client to authenticate the sender of a particular message. It can also be desirable to ensure that the message was not tampered with during transit. Our Technique describes a way for servers and clients to simultaneously add authentication and message integrity to HTTP messages by using a digital signature. HTTP Authentication defines Basic and Digest authentication mechanisms, TLS 1.2 defines cryptographically strong transport layer security, and OAuth provides a fully- specified alternative for authorization of web service requests. Each of these approaches is employed on the Internet today with varying degrees of protection. However, none of these schemes are designed to *cryptographically* sign the HTTP messages themselves, which is required to ensure *end-to-end message integrity*. An added benefit of signing the HTTP message for end-to-end message integrity is that the client can be authenticated using the same mechanism without the need for multiple round-trips.

Several web service providers have invented their own schemes for signing HTTP messages, but to date, none have been standardized. While there are no techniques in this proposal that are novel beyond the previous art, it is useful to standardize a simple and cryptographically strong mechanism for digitally signing HTTP messages. Our implementation provides a standardized implementation of the HTTP Signature Authentication Technique, specified in Oracle Drafts by M. Cavage et al.

Chapter 2

Prior Work

2.1 Basic HTTP Authentication

The first work towards securing HTTP messages as specified in [1] from 1999 which has now developed into Basic HTTP Authentication Scheme as specified in [2] from 2015. In the Basic authentication scheme, the HTTP user agent i.e. the client (e.g. a web browser or a banking portal, etc.) provides a user name and password when making a request. The HTTP request sent by the client contains a header field in the form of Authorization: Basic ¡credentials¿, where credentials are the Base64 encoding of username or user-ID and password joined by a single colon. This request when received by the server (e.g. a database provider or a service provider or a bank etc.) is authenticated with user credentials from the server database. And depending on this, the client is either authenticated or not. The HTTP Message passing between server-client includes the challenge and response flow works like this:

- ▷ The server responds to a client with a 401 (Unauthorized) response status and provides information on how to authorize with a WWW-Authenticate response header containing at least one challenge.

- ▷ A client that wants to authenticate itself with the server can then do so by including an Authorization request header with the credentials.

- ▷ Usually a client will present a password prompt to the user and will then issue the request including the correct Authorization header.

Counting towards its advantages, HTTP Basic authentication (BA) implementation offers the simplest technique for enforcing access controls to web resources because it does

not require cookies, session identifiers, or login pages; rather, the scheme uses standard fields in the HTTP header.

But the Basic Authentication mechanism does not provide confidentiality protection for the transmitted credentials since it involves shared secrets. They are merely Base64 encoded in transit and neither are cryptographically encrypted with any keys and algorithms nor hashed in any way. In simple words, it means that the username and password are passed in the clear with every request. Sending shared secrets in a non-encrypted manner makes the scheme more prone to replay attacks i.e. eavesdrop from hackers and attackers. This includes very subtle 401 phishing attacks in the form of slightly different authentication prompts (created by attackers) since the end process is user-dependent, it may introduce human errors in that form as well. Apart from this, the scheme also requires multiple rounds for authentication. The scheme also depends on the client to provide the authentication user experience, which needs to be well tested with the clients using the application. To overcome these advantages, a few new techniques are developed. One such technique is Digest Authorization, which is somewhat better in the sense that it sends an MD5 digest of some various bits including the username, password, and a nonce (among other values) and not the clear-text credentials. A password cannot be extracted from a captured digest. But again this results in an increase in the number of rounds for the authentication to complete.

2.2 TLS: Transport Layer Security

Another step towards providing security for HTTP transmission is Transport Layer Security (TLS) [3, 4, 5, 6], the successor of the now-deprecated Secure Sockets Layer (SSL). TLS is a cryptographic protocol designed to provide communications security over a computer network. The protocol is widely used in applications such as email, instant messaging, and voice over IP, but its use as the Security layer in HTTPS remains the most publicly visible.

The TLS protocol aims primarily to provide privacy and data integrity between two or more communicating computer applications. It runs in the application layer of the Internet and is composed of two layers: the TLS record and the TLS handshake protocols.

For a website or application to use TLS, it *must have an SSL certificate* installed on its origin server. A TLS certificate is issued by a certificate authority to the person or business that owns a domain. The certificate contains important information about who owns the domain, along with the server's public key, both of which are important for validating the server's identity.

A TLS connection is initiated using a sequence known as the TLS handshake. When a user navigates to a website that uses TLS, the TLS handshake begins between the user's device (also known as the client device) and the web server. This handshake (as the name suggests) is how the client and server 'talk turkey'. During this step – which happens in the blink of an eye – several things occur: the two entities kick it off by swapping messages. This step is necessary to confirm that both parties say they are who they are. Next comes the verification stage, followed closely by a little 'chat' about what type of encryption algorithms this type of secure communication will be using. The chat is concluded when both parties agree on session keys. This would be the TSL handshake at a glance.

But TLS has its disadvantages as well. Higher latency compared to other secure encryption protocols. A StackPath study revealed that connections encrypted by TSL have a 5ms latency compared to those that have not been encrypted. So, Whenever TLS is used, additional latency will be added to the site's traffic. Secondly, TLS is vulnerable to Man-in-the-Middle attacks, since the SSL certificate is sent along with. Besides this, other forms of cyber attacks such as SLOTH, POODLE, and DROWN are also found to be susceptible. Few platforms support TLS 1.3. There are a handful of platforms that support the latest TLS version: Chrome (version 67+), Firefox (version 61+), and Apple's Mac OS 10.3 (iOS 11). Popular Operating system provider Microsoft is facing difficulty with the implementation process.

Furthermore, implementing a TLS certificate isn't free, there are some costs involved.

Complexity in the network architecture is another major disadvantage of the TLS certificate. In this case, automatically the network topology can also become complex leading to more fail overs. For handling this there should be a proper network expert hired.

2.3 OAuth

The latest and most used technique for HTTP Authentication, typically among e-commerce vendors is OAuth [7, 8, 9]. OAuth stands for Open Authorization Framework and is the industry-standard delegation protocol for authorization. OAuth 2.0 is widely used by applications (e.g. SaaS platforms) to access user's data that is already on the Internet. That includes for example a user's contacts list on Google, user's friends list on Facebook, etc.

OAuth includes 4 actors in the process of access delegation:

- ▷ Resource owner (basically a user who has some private resources like email, photos,

etc.),

- ▷ Client (usually an application that wants to access these resources),
- ▷ Authorization server (who asks the resource owner for access to the resources on behalf of the client),
- ▷ Resource server (who stores user's private resources and shares them with authorized clients).

In some cases, the same application acts as both, the authorization server and resource server (e.g. Facebook). There are four flows (called grant types) to obtain the resource owner's permission (technically called access token): authorization code, implicit, resource owner password credentials, and client credentials.

Despite the remarkable use of OAuth across multiple platforms, there are still some areas where OAuth fails to impress.

1. There is no common format, as a result, each service requires its own implementation.
2. In the process of user verification, sometimes you have to make additional requests to get minimal user information. It can be solved with the help of the jwt token, but not all services support it.
3. When a token is stolen, an attacker gains access to the secure data for a while. To minimize this risk a token with a signature can be used.
4. Unnecessary complexity to support different services (Google, Outlook, Facebook, etc.)

Chapter 3

Our Work

3.1 Objective

As mentioned above as well, several web service providers have invented their own schemes for signing HTTP messages, but to date, none have been standardized. While there are no techniques in this proposal that are novel beyond the previous art, it is useful to standardize a simple and cryptographically strong mechanism for digitally signing HTTP messages. Our implementation provides a standardized implementation of the HTTP Signature Authentication Technique, specified in Oracle Drafts by M. Cavage et al.

The Broader Problem addressed by our implementation is to (1) Authenticate the sender of a particular message, (2) Keeping the identity secure, and in the process (3) minimize the Transmission Errors in the message.

What is required is a technique to deal with the confluence of all the three i.e. simultaneously: authenticate the sender by using a digital signature with no shared secrets and to also check message integrity.

3.2 Our Contribution

Our implementation provides a standardized implementation of the HTTP Signature Authentication Technique, specified in Oracle Drafts by M. Cavage et al.

We implement a "Signature" scheme which is intended primarily to allow a sender to assert the contents of the message sent are correct and have not been altered during transmission or storage in a way that alters the meaning expressed in the original message as signed. Any party reading the message (the verifier) may independently confirm the validity of the message signature.

This scheme is agnostic to the client/server direction and can be used to verify the contents of either HTTP requests, HTTP responses, or both.

3.3 Our Technique

Due to confidentiality constraints we are unable to fully specify the techniques employed by us for the HTTP Signature Authentication Scheme employed. In a nutshell, we deployed the following classes and techniques for our process:

▷ **Sender:**

- Signs specified HTTP Headers with Private Key.
- Sends the encoded sign along as a part of Signature in an Authorization Header.

▷ **Receiver:**

- Recreates Signature String [unsigned] from HTTP message.
- Verifies the recreated signature with original Signature using Public Key.
- Receiver's End API to Verify incoming messages
- Performs Authentication
- Ensures Message Integrity

▷ **Key Loader:** Loads the keys from different file formats.

▷ **Signature:** Defines the parameters and the construction of a signature object.

3.4 Our Results

Our implementation achieves the following benchmarks:

1. **No Shared Secrets for authentication:** We overcome the basic drawback faced by prior techniques by not allowing any shared keys or secrets between the server and the client.

2. **Simple to Implement:** We create a java library that now can be utilized as a wrapper on HTTP protocol to use our technique with a simpler and easier complexity. This makes our approach easily scalable and extensible to multiple user platforms.

3. **Can Ensure end-to-end message integrity:** Apart from authentication, we also achieve message integrity in our implementation.

4. **No need for multiple rounds or even a single round-trip:** Unlike Basic Authentication or TLS or OAuth, our implementation does not require any handshakes or middle-men or services to provide authentication.

5. **Agnostic to client/server direction:** The approach is agnostic to the server or client direction, anybody can act as a sender or verifier.

6. **Eligible for both HTTP requests and responses:** The process is also agnostic of the type of HTTP messages sent.

7. **Provides a standardized implementation of the HTTP Signature Authentication:** We provide a standardized implementation of the scheme proposed by Oracle draft by Cavage et al. which can now be deployed on Oracle Web Services.

3.5 Software Development Cycle and Technologies Used

While developing the concerned libraries for the HTTP Signature Authentication Scheme, we followed and achieved the following milestones:

1. **Literature Survey:** Studied Oracle Draft on Signing HTTP Messages by M. Cavage et al. Investigated various prior works and public key infrastructure and cryptography techniques to encrypt and decrypt messages.

2. **Orapki:** Learnt orapki utility by Oracle which is used for managing public key infrastructure (PKI) elements.

3. **Java Cryptography Architecture:** Mastered Java Cryptography Architecture (JCA) implementation which is a framework for working with cryptography using the Java programming language. This framework is essentially used by us in the major functions related to key-generation, reading and storage, and encrypting and decrypting messages using keys on the client as well as server end.

4. **Bitbucket**: BitBucket is an Oracle-based Git repository management solution. Setup a BitBucket repository for management and future scalability of the project with Oracle Solutions.

5. **Eclipse Java IDE:** We employed Eclipse Java IDE for the offline implementation of the project on our personal machine. Setup the directory structure for the java library.

6. **Main Implementation:** Because of confidentiality constraints we are unable to fully specify this step. In a nutshell, we employed a five-step development as follows:

   (a) **Key Management:** Created utilities to read and process different types of keys (public/private in multiple formats available.)

   (b) **Signature:** Created signature class and required functionalities for our implementation.

   (c) **HTTP Signer:** Created the classes corresponding to a signer, employing signature class to sign and send the corresponding HTTP messages from the client side.

   (d) **HTTP Verifier:** Created the classes corresponding to a verifier, employing signature class to receive and verify the corresponding HTTP messages at the server side.

   (e) **Demo Class:** Implemented a demo class to provide a working demo of the whole HTTP Signature Authentication Mechanism.

7. **Junit5:** Pursued the technique of JUnit5 for Unit Testing in Java and to create an up-to-date foundation for developer-side testing on the JVM. Tested a few programs such as Hello World, a program with Annotations, one with Annotations with Expected Exceptions, and another with Test Suite and Parameterized Tests.

8. **Documentation (OraDocs):** We used confluence oradocs to setup a documentation for our project. Updated the following in the project doc:

   ▷ Requirements of the project

   ▷ UML- Activity Diagrams corresponding to:

   - Signature String Construction
   - Creating the Signature
   - Verifying the Signature

Chapter 4

Conclusion and Future Direction

4.1 Conclusion and Significance of Our Work

By providing a standardized java-library implementation of the HTTP Signature Authentication Technique, specified in Oracle Drafts by M. Cavage et al., we introduce the "Signature" scheme which is intended primarily to allow a sender to assert the contents of the message sent are correct and have not been altered during transmission or storage in a way that alters the meaning expressed in the original message as signed. This allowed us to tackle the broader problem of authenticating the sender of a particular message, meanwhile keeping the identity secure, and in the process minimizing the transmission errors in the message.

Besides this, the implementation also ensures that there are No Shared Secrets for authentication and that the library rendered is simple and easy to implement for further usage. We also diminish the need for multiple rounds or even a single round-trip for the authentication, which is in-fact agnostic of any type of HTTP messages sent.

4.2 Future Directions

The work can be further extended and improved upon using the following pointers:

1. **Python API:** For now, our work is limited to employing Java Cryptography Architecture to provide Java Libraries for the same. But the same approach can be extended across multiple languages such as Python, C/C++, Ruby as well.

2. **Key Agnostic:** A major portion of our implementation included reading, storing, and managing public and private keys in various formats. Due to the multiplicity of these key formats, it is essential to included and extend our approach to as

many types and formats of keys out there in the public key infrastructure to ensure scalability of the library.

3. **A Front-End:** For programmers and developers, a simple java-library which we implement would be of the utmost usage. But one can also try to indeed create a debonair graphic user interface for the same.

References

[1] Professor John Franks, Phillip Hallam-Baker, Lawrence C. Stewart, Jeffery L. Hostetler, Scott Lawrence, Paul J. Leach, and Ari Luotonen. HTTP Authentication: Basic and Digest Access Authentication. RFC 2617, June 1999.

[2] Julian Reschke. The 'Basic' HTTP Authentication Scheme. RFC 7617, September 2015.

[3] Mishal Room. 5 Advantages and Disadvantages of TLS — Weaknesses & Benefits of TLS . https://www.hitechwhizz.com/2020/08/5-advantages-and-disadvantages-drawbacks-benefits-of-tls.html, 2018. [Online; accessed 19-July-2021].

[4] VLADIMIR UNTERFINGHER. What is Transport Layer Security (TLS)? Strengths and Vulnerabilities Explained . https://heimdalsecurity.com/blog/what-is-transport-layer-security/, 2020. [Online; accessed 19-July-2021].

[5] Wikipedia. Transport Layer Security. https://en.wikipedia.org/wiki/Transport_Layer_Security#Algorithms, 2021. [Online; accessed 19-July-2021].

[6] CloudFare. What is TLS (Transport Layer Security)? https://www.cloudflare.com/en-in/learning/ssl/transport-layer-security-tls/, 2018. [Online; accessed 19-July-2021].

[7] Agraj Mangal. What is OAuth? https://code.tutsplus.com/articles/oauth-20-the-good-the-bad-the-ugly--net-33216, 2020. [Online; accessed 19-July-2021].

[8] Damian Rusinek. OAuth. https://medium.com/securing/what-is-going-on-with-oauth-2-0-and-why-you-should-not-use-it-for-authentication-5f4
2018. [Online; accessed 19-July-2021].

[9] Oleksii Andrukhanenko. Oauth 2.0 Basic Understanding. https://stfalcon.com/en/blog/post/oauth-2.0, 2020. [Online; accessed 19-July-2021].