4-2019

# An Authentication mechanism for stateless communication

Ana Valentina Rodriguez Sosa
avr9503@rit.edu

# An Authentication mechanism for stateless communication

by

**Ana Valentina Rodriguez Sosa**

A Thesis Submitted
in
Partial Fulfillment of the
Requirements for the Degree of
Master of Science
in
Computer Science

Supervised by

Dr. Rajendra K. Raj
Dr. Carol Romanowski

Department of Computer Science

B. Thomas Golisano College of Computing and Information Sciences
Rochester Institute of Technology
Rochester, New York

April 2019

The thesis "An Authentication mechanism for stateless communications" by Ana Valentina Rodriguez Sosa has been examined and approved by the following Examination Committee:

Dr. Rajendra K. Raj
Professor
Thesis Committee Chair

Dr. Carol Romanowski
Professor
Thesis Committee Co-Chair

Dr. Sumita Mishra
Professor
Thesis Committee Observer

# Dedication

To my husband, Sebastian, who followed me to pursue this dream and he always believed in me.

# Acknowledgments

I want to thank you to my advisors, Professor Rajendra K. Raj, Professor Carol Romanowski and Professor Sumita Mishra who believed in me and make this happen.

Special thank you to Professor Carol Romanowski who support me and gave me insight observations in my report.

Cindy Wolfer who was always there for me when I needed, gave me advice and support.

A special thanks to my husband, Sebastian. It is difficult to put in words what meant to me having you here with me every day, and I know I can always count with you.

Thank you to my husband cousin, Professor Marcelo Ponce, who was always there for me giving support and advice in this challenging phase of my life.

Thanks to RIT for giving me this tremendous opportunity of having this dream come true.

# Abstract

**An Authentication mechanism for stateless communication**

**Ana Valentina Rodriguez Sosa**

**Supervising Professors: Dr. Rajendra K. Raj**
**Dr. Carol Romanowski**
**Professor**
**Thesis Committee Co-Chair**

Most of the applications we use on a daily basis are distributed systems that are composed of at least one client and server and are exposed to the Internet. This communication is based on an HTTP protocol, which is a stateless protocol. Because of its communication characteristics, developers are forced to implement a series of mechanisms to pursue user privacy, security as well as business features. Modern social media applications such as Facebook have been using secure tokens as an authentication mechanism. These applications are relying on only one part of the approach, such as token mechanism generation. If the third party system does not consider another aspect of security, the authentication mechanism will fail unless we consider all the aspects in the user authentication process, as shown when Facebook shared private user tokens with unauthorized users. More than 50 million accounts were affected, and another 40 million could be affected as well. This work introduces a secure mechanism to identify the user in an enterprise/web application across all user interactions once the user has logged in. The system to be proposed creates a relationship between the user and the session management for each system. This project aims to show a different perspective based on a user-centered approach, where the approach is based on the user and its user access and not only on an ID/Token mechanism. The research proposes that the session manager mechanism can be more secure as well as the token-based mechanism. The approach integrates Blockchain technology for representing the relationship between the user and a system.

# Contents

# List of Figures

# Chapter 1

# Introduction

Nowadays, Web and Enterprise applications are growing every day and protecting data in a non-trusted environment represents a challenge, especially in a non-secure and stateless environment. User authentication has become one of the most critical aspects in security and software development. The user's state helps the system to identify him and then enables characteristics such as access control, localization, and so on. Owners of applications are responsible for protecting user information and any other data from unintended access or malicious attempts. Users preferred to have an identified user state to take advantage of enhanced user experiences, request unique user data, and access data which is related to their company and role. To accomplish these requirements and being able to protect user information, user login functionality has been developed as part of the system development. At first glance, this may look simple, but as we know, web systems use HTTP protocol for exchanging data between the user and the system. The HTTP protocol is stateless, which means that data exchanged between the user and the system does not have historical information to identify or track the user through his multiple requests. How the systems can maintain the user state over the stateless communication without compromising user experience and security? Here is where sessions and tokens come into play. Both allow systems to track user actions over the system. Session management provides capabilities for maintaining user state based on the session ID. The Web Application Security Project's (OWASP) defines the top 10 software vulnerabilities. [13] Broken authentication has been part of this list for several years and it has been in the second place in the last

4 years. Broken authentication includes Session Management and Authentication mechanisms. The current situation is that while most modern social media applications such as Facebook have been relying on token authentication mechanisms, these mechanisms are third-party solutions and are vulnerable if are shared with non-authorized users. In the famous Facebook incident, users' private tokens were shared with unauthorized users, exposing as many as 90 million accounts [17]. Thinking in new ways of protecting user data and creating out-of-the-box solutions could help to change the actual status of affairs. Hence, then to mitigate the risk of broken authorization, the aim of this work is to present a more secure session management mechanism by integrating Blockchain technology.

# Chapter 2

# Background

## 2.1 An Overview of Authentication mechanisms

Authentication is a security mechanism that allows systems to identify the user as a registered user by proving information to proof the user is who he/she claims to be. There are different authentication mechanisms based on biometrics, usernames-password, certificates, tokens, etc. One of the most common mechanism is the combination of username and password. However, having this approach isolated from other security issues has lead to many attacks, forcing security experts and developers to find other robust and enhanced mechanisms. Some alternatives are HTTP-based authentication (basic, digest) by using HTTP headers. Other modern approaches have been implemented such as two-factor authentication, or password-less mechanisms.

## 2.2 Authentication Mechanisms for Login

There are different mechanisms for authenticating a user to login into a application.

**Password protection** Passwords can be protected by using hashing and salting. Salting and hashing can be used together for avoiding password visibility. Salting generates a random value to be applied to the password. Hashing obscures the password by transforming it something indecipherable; this value will be stored into the database, making it practically impossible to use for future intruders.

**Biometrics** Authentication by biometrics verification is widely used nowadays by different systems. Some of the options available are fingerprinting, face recognition, voice waveform recognition, retina and iris patterns. Researchers created patents based on identifying user behavior patterns associated with biometrics. Some of these behaviors are based on common patterns found in users that are related to movement and posture, such as movement of the mouse, eye movement and so on.

**WebAuthn: Password-Free** FIDO (Fast Identity Online) is a mechanism that allows users to log in into systems without the need of user and password credentials. FIDO was developed by companies that are part of the World Wide Web Consortium (WC3). The main idea is to improve the user experience and create a robust and secure mechanism for authenticating users in a system. FIDO is using the pre-existent specifications: FIDO (Fast Identity Online), Universal 2nd Factor Authentication (U2F) and Universal Authentication Framework (UAF) for verifying user identities. To login into the system the user provides password information and as a second authentication mechanism biometrics or other mechanisms.

## 2.3 Authentication mechanisms for maintaining user state

There are different mechanisms developed through the years to maintain the user's state in a web/enterprise system after the user has logged into the application. Some of these methods are based on session management, token mechanisms, or a mix of both.

### 2.3.1 Systems implementing Token-based Mechanisms

**OAuth 2.0** is a token-based mechanism used by companies around the world, such as Google, Facebook, Twitter, LinkedIn [7]. It provides capabilities for user authorization, but it cannot be defined as an authentication protocol itself. As developers we cannot rely on it since OAuth does not know who the user is, if there is a user or not. According to an OAuth advocate, ”This has led many developers and API providers to incorrectly conclude that OAuth is itself an authentication protocol and to mistakenly use it as such. Let's say that again, to be clear" [14].

**OpenID Connect** is a standard for authenticating user access in different systems. This mechanism is based on OAuth2.0 and it communicates if the user has access to the third party without revealing user information such as date of birth, files, name and other data. This is rephrased in the OpenID Connect website as (Identity, Authentication) + OAuth 2.0 = OpenID Connect[15].
OpenID connects a new type of token to OAuth 2.0, enabling the Relying Party (RP) to verify user identity. The token will be used as a complementary access token as part of OAuth 2.0. The token ID contains claims about the user authentication such as:

- The identity of the OpenID provider (OP)
- The user's unique identifier at this OP
- The identification of the intended recipient
- Issued time
- Expired time

The data is parsed and converted into a JSON Web Token and then digitally signed by the OP. The user credentials are part of the token. This information is used as a user authorization in the resources protected by a third party. An API verifies both of these tokens.

**Security Assertion Markup Language Tokens (SAML)** is an open standard for authentication and provides secure data exchange. It supports Single Sign On (SSO) by transferring the user information to another third-party through the exchange of a signed XML file [1].

**Simple Web Tokens (SWT)** is based on key-value pair encoded in HTML. The result values are verified by SHA 256 HMAC using a public key.

**JSON Web Tokens (JWT)** could be used as an authentication mechanism as well as a secured communication exchange. The information sent is digitally signed by using a private/public key. Each token can be encrypted. Also, JWT tokens are less verbose than SAML [3] [11]. JWT implements a standard, RFC 7519.

### 2.3.2 Session Based Mechanisms

A session is created once a user logs into the system successfully. The server creates a session object for keeping track of each user during the requests and responses, and to identify the user during the system session life.

The session object is stored on the client side and is maintained on the server side. In the first response, after the user logs in, the server sends the session information in the HTTP header. Then, the client sends this information in each request to the server for authentication and tracking purposes. The server validates the session information to identify the user. The session management creates a temporary authentication mechanism for the user's interactions.

In general, on the client side, the session information is stored in a cookie [16]. However, there are other storage mechanisms:

- URL parameters
- Hidden form fields

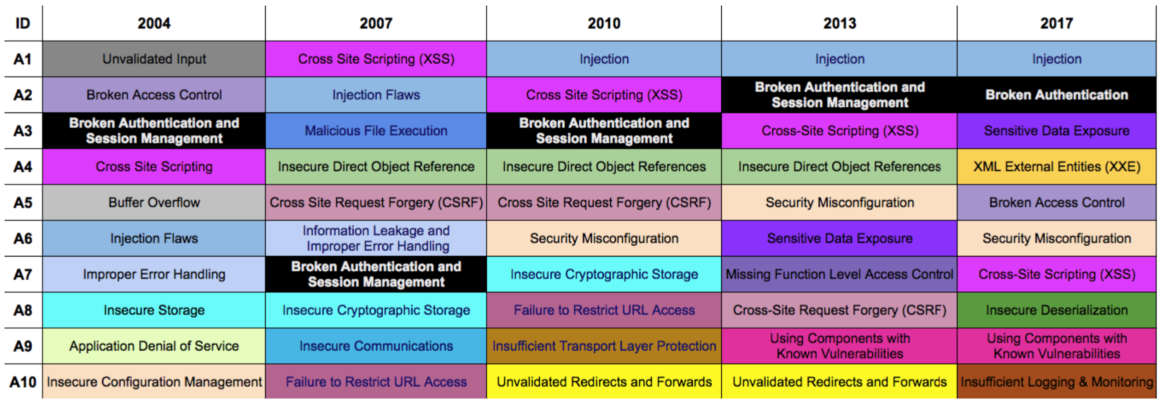
## 2.4 Related work

In my review of the literature, I did not find any approach for protecting the session ID. Different approaches are available to protect tokens, some of which are implemented in Blockchain, while others use Blockchain only for maintain user identity.

There are several systems based on identity management. Some of them are implemented on Blockchain, some are mobile apps and others focused on attestation attributes. Most of these methods are trying to create a unique system that resolves everything, not only authentication but identity management as well. But, this approach could be controversial since creating a unique solution may limit systems and users to only one solution instead of providing flexibility and the freedom for choosing and and using different platforms [8].

### 2.4.1 The Problem

Session management and broken authentication problems have been eyed by OWASP since the OWASP Top 10 Vulnerabilities have been published. Figure 2.1, shows the OWASP Top 10 through the years. Broken Authentication and Session management (highlighted in the table) started in third place in 2004, and have always been part of the OWASP Top 10. Broken Authentication, 13 years later, is still in the top positions of the list, topping even the exposure of sensitive data.



| ID | 2004 | 2007 | 2010 | 2013 | 2017 |
|---|---|---|---|---|---|
| A1 | Unvalidated Input | Cross Site Scripting (XSS) | Injection | Injection | Injection |
| A2 | Broken Access Control | Injection Flaws | Cross Site Scripting (XSS) | **Broken Authentication and Session Management** | **Broken Authentication** |
| A3 | **Broken Authentication and Session Management** | Malicious File Execution | **Broken Authentication and Session Management** | Cross-Site Scripting (XSS) | Sensitive Data Exposure |
| A4 | Cross Site Scripting | Insecure Direct Object Reference | Insecure Direct Object References | Insecure Direct Object References | XML External Entities (XXE) |
| A5 | Buffer Overflow | Cross Site Request Forgery (CSRF) | Cross Site Request Forgery (CSRF) | Security Misconfiguration | Broken Access Control |
| A6 | Injection Flaws | Information Leakage and Improper Error Handling | Security Misconfiguration | Sensitive Data Exposure | Security Misconfiguration |
| A7 | Improper Error Handling | **Broken Authentication and Session Management** | Insecure Cryptographic Storage | Missing Function Level Access Control | Cross-Site Scripting (XSS) |
| A8 | Insecure Storage | Insecure Cryptographic Storage | Failure to Restrict URL Access | Cross-Site Request Forgery (CSRF) | Insecure Deserialization |
| A9 | Application Denial of Service | Insecure Communications | Insufficient Transport Layer Protection | Using Components with Known Vulnerabilities | Using Components with Known Vulnerabilities |
| A10 | Insecure Configuration Management | Failure to Restrict URL Access | Unvalidated Redirects and Forwards | Unvalidated Redirects and Forwards | Insufficient Logging & Monitoring |

Figure 2.1: Analysis of Vulnerabilities changes over time-based on the OWASP Top Vulnerabilities since 2003[13]. Each vulnerability has a unique color to make visible its evolution through the years.

#### Session Management issues

There are potential threats and attacks related to session management:

- Session Hijacking: is the process when a hacker steals or predicts the session Id. Since session Ids are usually recorded in cookies as part of the client and server communication, the hacker can use the stolen information to access the system as an authenticated user. Some of techniques used to commit this attack are Cross-site Scripting (XSS), Man In The Middle attack (MITM), Man In The Browser attack (MITB,MitB,MIB, MiB), and session fixation.

To avoid Session Hijacking, a new cookie should be generated each time the user begins a new session and communications should be encrypted using Hypertext Transfer Protocol Secure(HTTPS) [4].

HTTPS implements an exchange of certificates issued and verified by a trusted certificate authority (CA) hence making this a mutual authentication. This communication will validate that the messages sent are from the owner's certificates [19].

- Session Prediction: Hackers can easily predict session ids and tokens if the systems do not create robust and secure session Ids [16]. There are several recommendations for making session Ids indecipherable, including creating an ID that contains an extremely large number of characters and also by using pseudo-randomness. Robust ids can be created using all possible combinations of attributes:

  - IP address
  - User-Agent
  - Request timestamp
  - A random number
  - A secret key and SHA-256
  - Timestamp
  - XOR obfuscation

  Other ways to protect session Ids include labeling cookies as secure, avoiding URL queries to transmit tokens, and using a post method with hidden field values. These will decrease the possibility that the session ID will be visible to the attacker. Implementing logouts, session timeouts, and two-step verification will reduce the possibility of accessing as an authenticated user when the user is not longer active in the application.

- Session Fixation: This type of attack is similar to the one described in Session Hijacking, but takes place before the user login. The attacker tries to gain a valid session ID by creating a session ID and tricking the user with un-trusted information to log into the website. This attack is related to session prediction. A common technique used in Session Fixation is to generate a cookie on the client side and then hide the new session ID in a field using a form made by the attacker, or through a URL sent to the user [9].

### Tokens issues

Token mechanisms are undoubtedly robust and very popular in the software industry. Most of these systems are focused on making the token indecipherable. However, there are aspects of security that are not addressed in current approaches. One of those aspects is token exposure. In the last data breach at Facebook, September 2018, the tokens were exposed in the GUI by a third-party library, making those tokens visible to other users. The tokens were expropriated by other parties, making it possible to impersonate an authenticated user and access the system. Once an intruder accessed the system with the stolen token, the user's content will be visible as well as credit card information and other private information. Facebook was forced to generate more than 50 million new tokens [17].

This analysis shows that Broken Authentication is a critical flaw in the eyes of the security community based on data breaches and impact in the software industry. There are several potential threats and attacks related to session management, occurring at different points in the session timeline and from different aspects of the process.

## 2.5 Hypothesis and Methodology

Because of the importance of session management, this element has become a target for hackers. If hackers access a user's session id, they could impersonate an admin user and gain access and control of the whole system without having been authenticated. A hacker could access the system despite authentication robustness if session management is not part of this authentication mechanism.

### 2.5.1 Hypothesis

Nowadays some web and enterprise applications are relying completely on tokens, or Session ID which allows hackers to access the application without the need to login. We can change this approach by creating a strong session management system that does not only rely upon a third-party mechanism, and also in the user, to prove the user's authenticity. This new approach based on session management and the user creates a strong relationship between user identification and session IDs across different systems in a decentralized system. The hypothesis is that this new approach will reduce session management broken authentication.

### 2.5.2 Technical Approach: Blockchain

Blockchain can be defined as a distributed database mechanism which provides scalability, security, and privacy. Satoshi Nakamoto [10] started the Blockchain, as a revolutionary idea when he wrote a paper about it in 2008. After this period, Blockchain has become a source of new ideas; creating new paradigms, innovative approaches and creating a path for resolving real-world problems differently. Blockchain has received more than 1B USD of investment, and more than 12.4M wallets were created in 2014. Several Blockchain systems have been developed since then, in different areas such as financial, health, e-commerce, cloud and one of the most important, for rewriting the internet[2]. There are companies providing huge funding for rewriting social media apps to be allocated in the Blockchain Network [2]. Blockchain addresses and resolves many issues around security and privacy, creating the opportunity and vision for better security approaches. The philosophy that runs behind it has been primarily embraced because Blockchain implements a P2P network without an intermediary, promoting a decentralized authority.

### Security

The *CIA triad* is a model designed for resolving and identifying issues related to information security. CIA is defined by the three concepts: Confidentiality, Integrity and Availability.

**Confidentiality** This property represents how the data is protected when a user intends to access the information. Blockchain implements confidentiality by creating transactions without revealing user information. However, the transaction could lead to the user who created the transaction. There are different approaches to resolve this issue, one of which uses synonymous addresses to avoid tracking the source.

**Integrity**

This property ensures that the data is trustworthy, accurate and consistent across the system. Blockchain implements integrity by creating immutable transactions that avoid data to be changed in a particular point of time. Also, it captures all data changes over the time in its immutable ledger.

**Availability** This property represents the ability of a system to be accessible at any time in a consistent way. Consistency is one of Blockchain's main characteristics. However, some recent research showed that Blockchain could not meet availability expectations due to slow responsiveness associated with committed transactions [18].

## 2.6 Roadmap

This thesis is organized as follow:

Chapter 2 gives a background information about authentication mechanisms, hypothesis and methodology.

Chapter 3 presents the technical approach and its implementation, the use cases for this research and how the data flow and interact between components, testing the smart contract is included based on the use cases created.

Chapter 4 presents an analysis based on doing penetration testing and vulnerabilities scan in a set of selected applications to validate the hypothesis.

Chapter 5 describes the research conclusions, current status, lessons learned and future work.

# Chapter 3

# Approach, Design and Implementation

This chapter describes the technical approach followed, its implementation and testing based on the use cases.

## 3.1 Design

The model presented is shown in Fig. 3.1 and described below:
-The user: Any user who interacts with the system.

-Key/wallet: Each registered user will have an authentication wallet. The digital wallet is representing the key identity for accessing different systems. All systems in which the user has been registered in any way will have a unique key. The wallet will store the key pairs public-private related to each system.

-Public/private key: the key generation will take place once the user is registered in the system. Each user will have his own wallet and keys. The user will sign each message using his private key, in this way we could ensure that the user is who we think it is; besides, by using Blockchain, the relationship between the user and session ID will be immutable.

-Authentication mechanism:

The login mechanisms are out of the scope in this research. However, in the implementation, we will need them to test and validate the model. The main idea is that the system could use its session ID (generated by the system) or use the session generated by an enterprise or web application.

-Session Management component:

This element is the gateway for creating a channel between the web application and the Blockchain. It will create a layer of abstraction and communication for generating data understandable for Blockchain and executing smart contract definitions. The system could be configured to use a proprietary session ID or a strong custom Id.

-Data Storage Components:

(a) Storage definitions: This component is in charge of generating the data for being stored in the Blockchain and manipulated through the smart contract.

(b) Data Storage Admin: This element represents the contract implementation to be deployed in the Blockchain. The contract will be in charge of creating the transactions for each user-application-session relationship, validating the transactions according to contract as well as invalidating the user session information.

-Blockchain- Ethereum:

The Blockchain will be in charge of attaining the relationship between the user-application-session. The Blockchain implementation will be Ethereum. To implement and test the system, use cases for relevant operations must be defined.

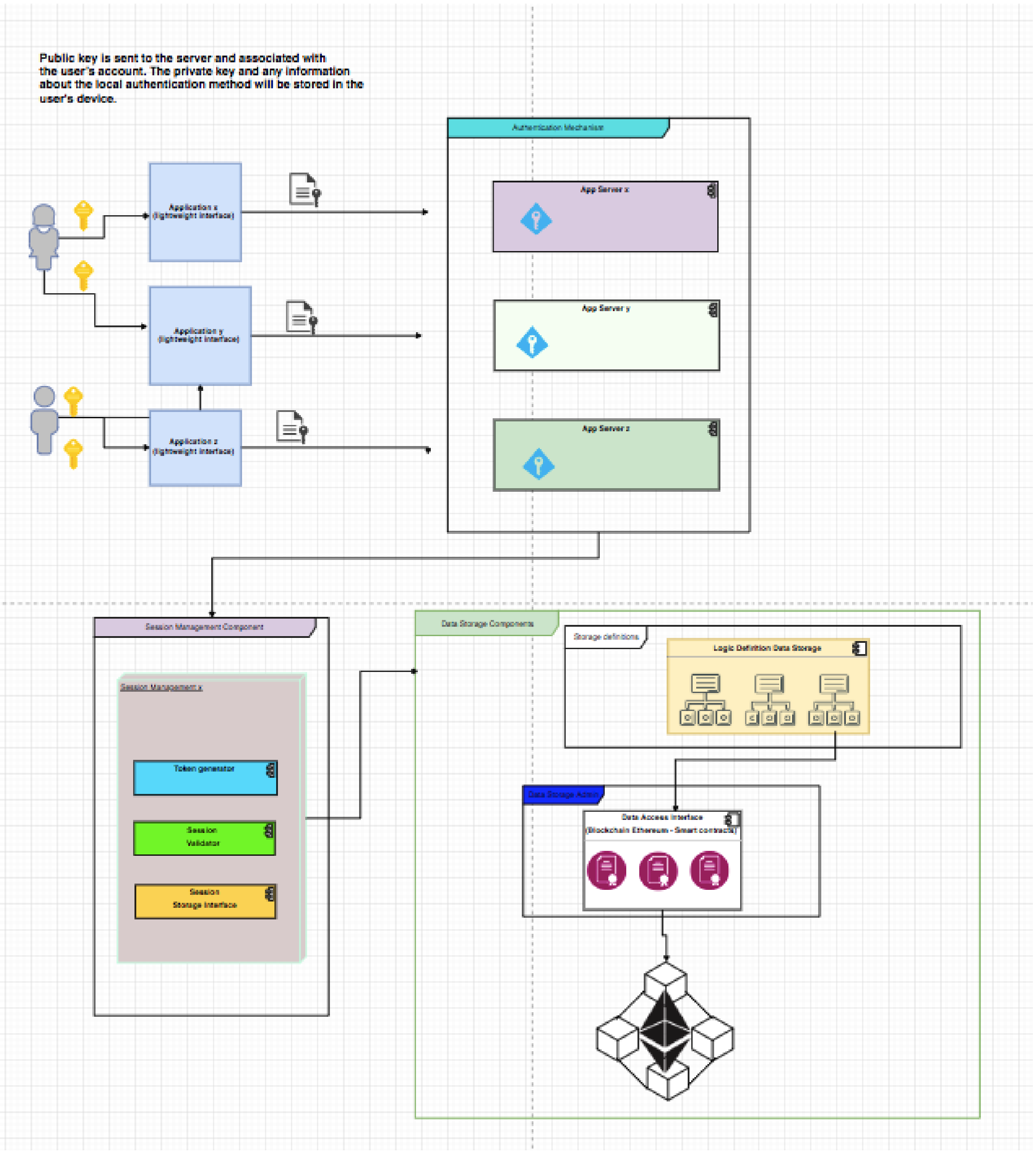

Public key is sent to the server and associated with the user's account. The private key and any information about the local authentication method will be stored in the user's device.
Authentication Mechanism
Application x (lightweight interface)
Application y (lightweight interface)
Application z (lightweight interface)
App Server x
App Server y
App Server z
Session Management Component
Session Management x
Token generator
Session Validator
Session Storage Interface
Data Storage Components
Storage definitions
Logic Definition Data Storage
Data Storage Admin
Data Access Interface (Blockchain Ethereum - Smart contracts)

Figure 3.1: A model for Authentication Mechanism for stateless Communication

## 3.2 Use cases

**Login** Here the user is attempting to login into the web/enterprise application. The application validates the user, but will ask the Blockchain to validate the session data and create the session for the user (Figure 3.2).

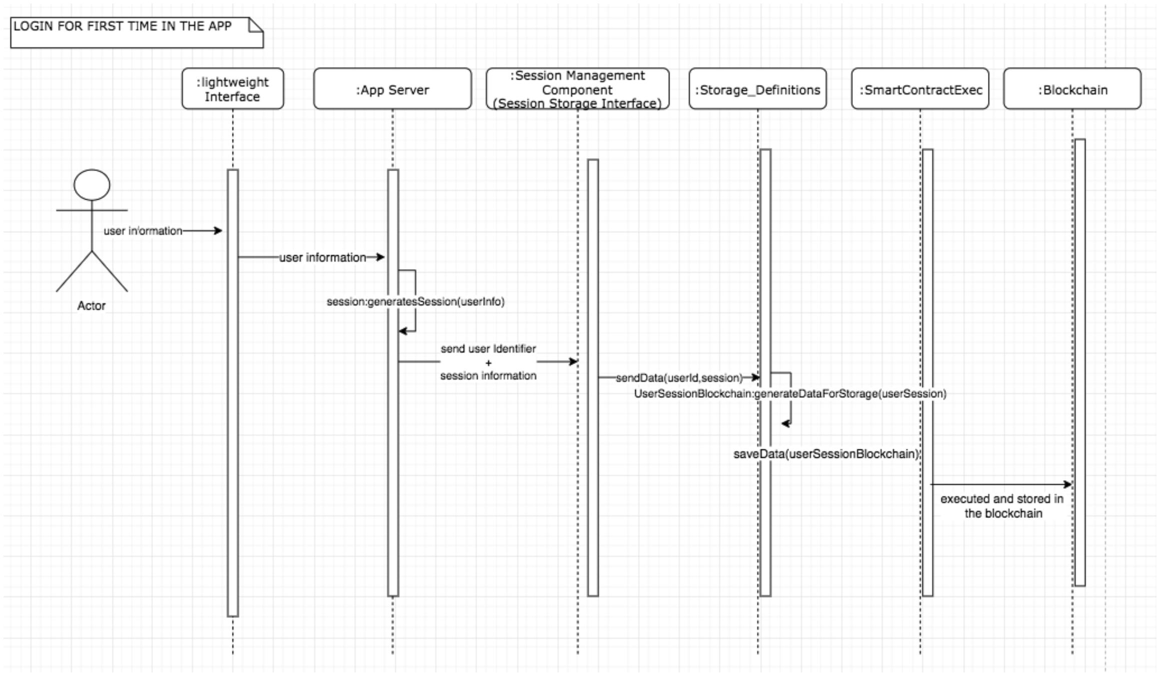

LOGIN FOR FIRST TIME IN THE APP
:lightweight Interface
:App Server
:Session Management Component (Session Storage Interface)
:Storage_Definitions
:SmartContractExec
:Blockchain
Actor
user information
user information
session:generatesSession(userInfo)
send user Identifier + session information
sendData(userId,session)
UserSessionBlockchain:generateDataForStorage(userSession)
saveData(userSessionBlockchain)
executed and stored in the blockchain

Figure 3.2: Flow for login use case

**Requests** In this use case, the user is sending a request to the web/enterprise application. The application validates the data; however, the Blockchain needs to validate that the session data is not null, is the same as the previous valid session data, and that the expiration date is still valid. Blockchain will validate the session dates, creation, and expiration versus the current date-time. If the session is no longer valid, the Blockchain will expire the user session ( Figure 3.3).

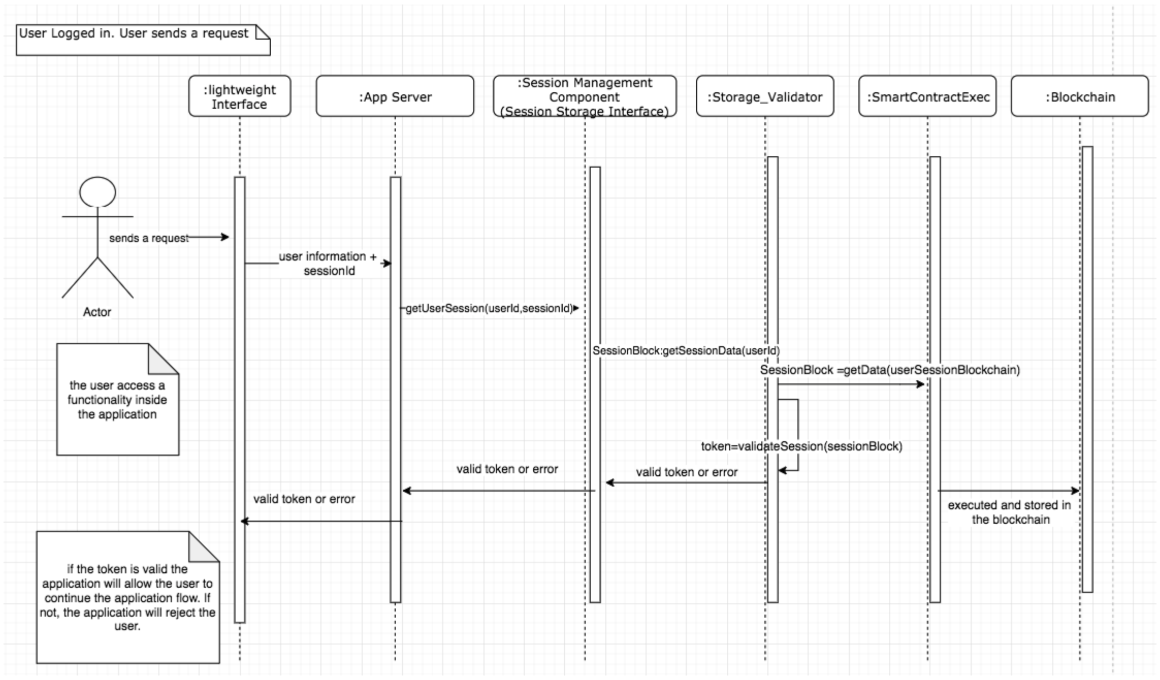

User Logged in. User sends a request
:lightweight Interface
:App Server
:Session Management Component (Session Storage Interface)
:Storage_Validator
:SmartContractExec
:Blockchain
sends a request
Actor
user information + sessionId
getUserSession(userId,sessionId)
SessionBlock:getSessionData(userId)
SessionBlock =getData(userSessionBlockchain)
the user access a functionality inside the application
token=validateSession(sessionBlock)
valid token or error
valid token or error
valid token or error
executed and stored in the blockchain
if the token is valid the application will allow the user to continue the application flow. If not, the application will reject the user.

Figure 3.3: Flow for any requests after login

**Logout** In this case, the user logs out from the web/enterprise application or the application decides to finish the user session due to time expiration, an idle session, or other session ending scenarios. The application is responsible to notify the Blockchain. However, the Blockchain will also be checking if the session is still valid in every request, and will expire the session if the session data is not valid ( Figure 3.4).

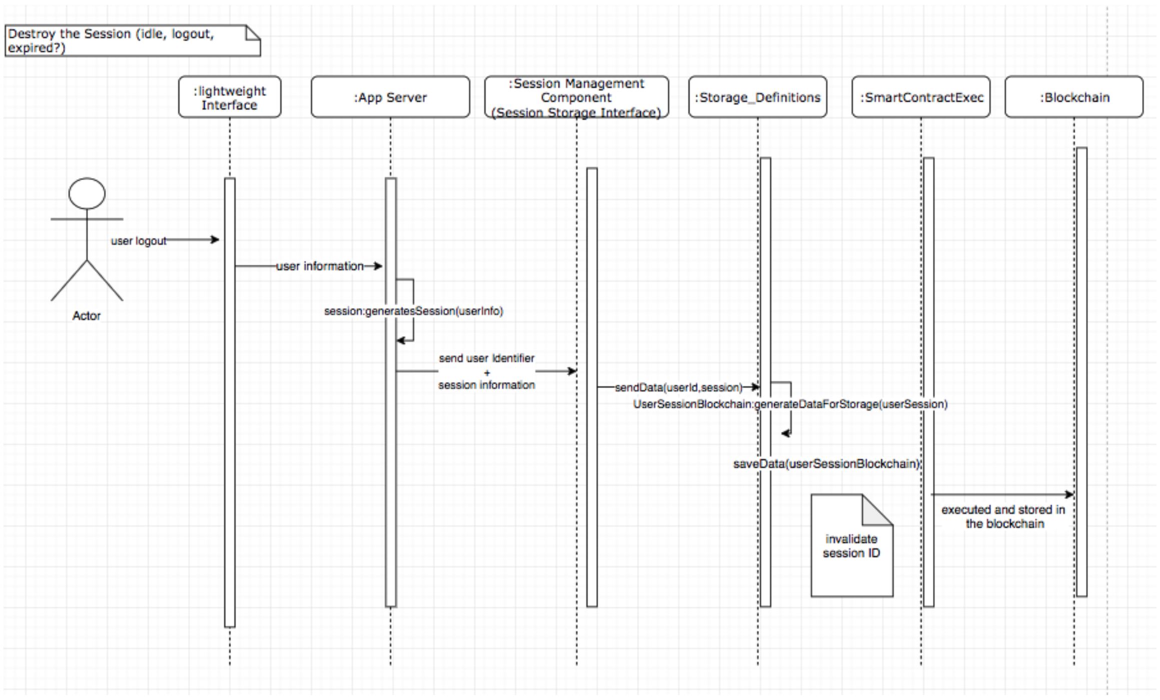

Destroy the Session (idle, logout, expired?)
:lightweight Interface
:App Server
:Session Management Component (Session Storage Interface)
:Storage_Definitions
:SmartContractExec
:Blockchain
Actor
user logout
user information
session:generatesSession(userInfo)
send user Identifier + session information
sendData(userId,session)
UserSessionBlockchain:generateDataForStorage(userSession)
saveData(userSessionBlockchain)
invalidate session ID
executed and stored in the blockchain

Figure 3.4: Flow for logout use case

## 3.3 Implementation

To test the hypothesis, two different applications were used:

(a) Session management applications, one based on Java as an Enterprise Edition and one based on Drupal as a Web application based on Php. (b) An approach based on Blockchain using smart contracts and Solidity language.

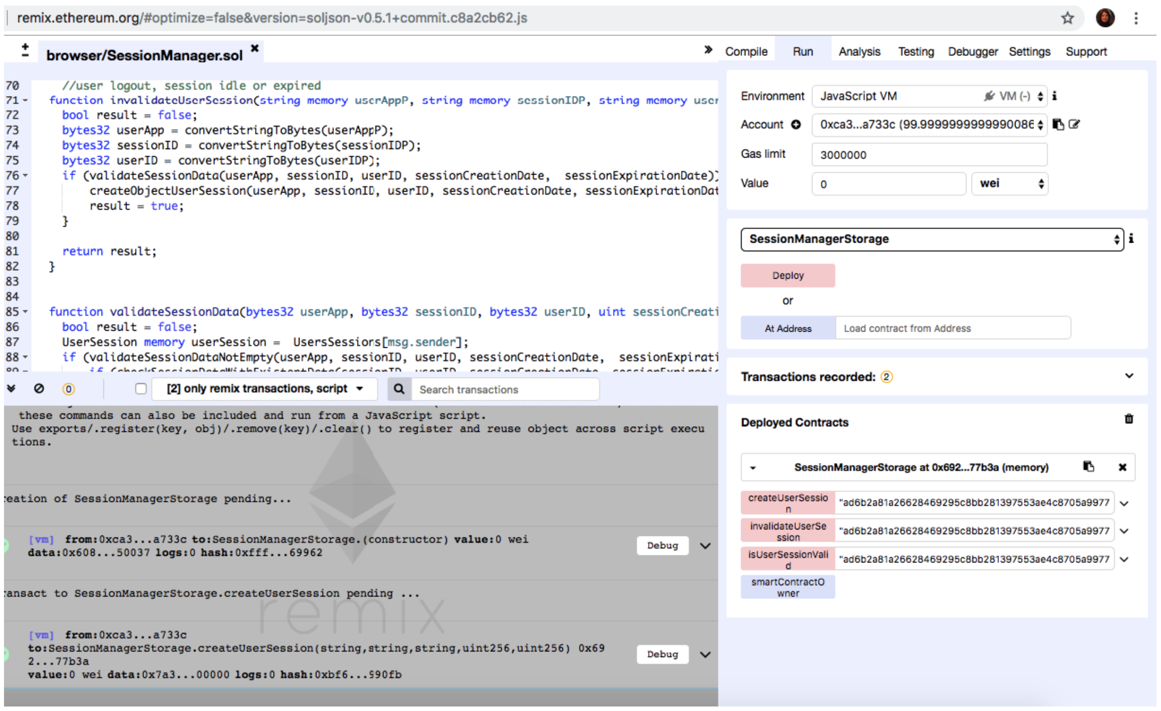
- Environment for Blockchain - Ethereum
  - Ethereum Remix
  - Deploy scripts
  - timestamp converter
  - bycrypt converter (for testing Java Enterprise)
- Environment for development - PHP Web

  - XAMPP - is a development suite tool containing Apache, MariaDB, PHP.
  - A website implemented in Drupal 8.3.7 was used as part of my experiment.

- Environment for development Java Enterprise
  - Spring Tool Suite
  - JDK 1.8
  - Spring Framework

## 3.4 Testing the smart contract

The smart contract was tested using Remix Ethereum website, Fig. 3.5. This tool provides a straightforward way for debugging and provides access to the Stack, Memory, Solidity State and local values.



```
  //user logout, session idle or expired
function invalidateUserSession(string memory userAppP, string memory sessionIDP, string memory user
  bool result = false;
  bytes32 userApp = convertStringToBytes(userAppP);
  bytes32 sessionID = convertStringToBytes(sessionIDP);
  bytes32 userID = convertStringToBytes(userIDP);
  if (validateSessionData(userApp, sessionID, userID, sessionCreationDate,  sessionExpirationDate))
      createObjectUserSession(userApp, sessionID, userID, sessionCreationDate, sessionExpirationDat
      result = true;
  }

  return result;
}


function validateSessionData(bytes32 userApp, bytes32 sessionID, bytes32 userID, uint sessionCreati
  bool result = false;
  UserSession memory userSession =  UsersSessions[msg.sender];
  if (validateSessionDataNotEmpty(userApp, sessionID, userID, sessionCreationDate,  sessionExpirati
```

```
 these commands can also be included and run from a JavaScript script.
Use exports/.register(key, obj)/.remove(key)/.clear() to register and reuse object across script execu
tions.

eation of SessionManagerStorage pending...

 [vm]  from:0xca3...a733c to:SessionManagerStorage.(constructor) value:0 wei
 data:0x608...50037 logs:0 hash:0xfff...69962

ansact to SessionManagerStorage.createUserSession pending ...

 [vm]  from:0xca3...a733c
 to:SessionManagerStorage.createUserSession(string,string,string,uint256,uint256) 0x69
 2...77b3a
 value:0 wei data:0x7a3...00000 logs:0 hash:0xbf6...990fb
```

Environment: JavaScript VM  VM (-)
Account: 0xca3...a733c (99.99999999999990086
Gas limit: 3000000
Value: 0 wei
SessionManagerStorage

createUserSession: "ad6b2a81a26628469295c8bb281397553ae4c8705a9977
invalidateUserSession: "ad6b2a81a26628469295c8bb281397553ae4c8705a9977
isUserSessionValid: "ad6b2a81a26628469295c8bb281397553ae4c8705a9977

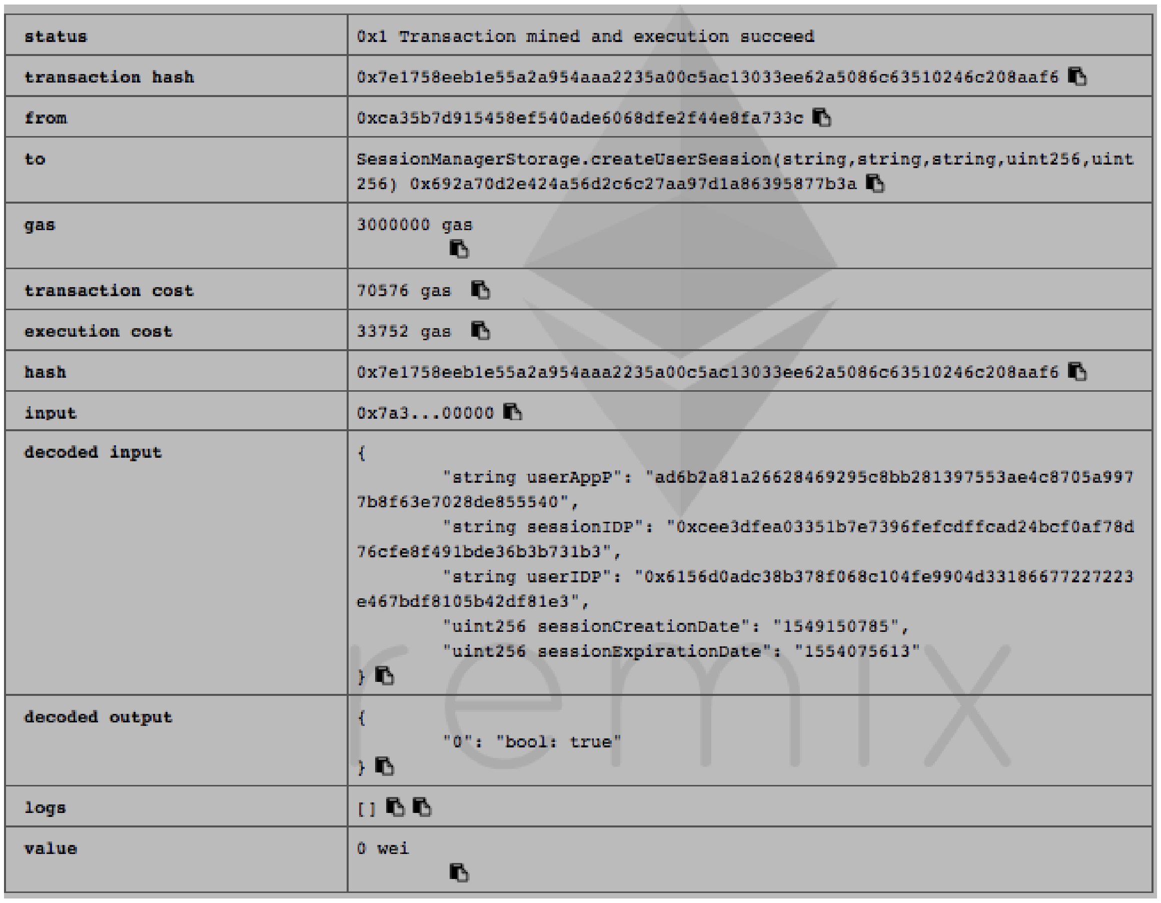
Figure 3.5: Remix Ethereum Testing Session Management Smart Contract

### 3.4.1 Use cases tested

I analyzed the possible scenarios, and I created use cases for each main functionality. The use cases described above:

#### Create a user session with valid data

The user is attempting to login into the web/enterprise application. Result: The Blockchain validates the data, creates the user session and returns Ok. Fig. 3.6.

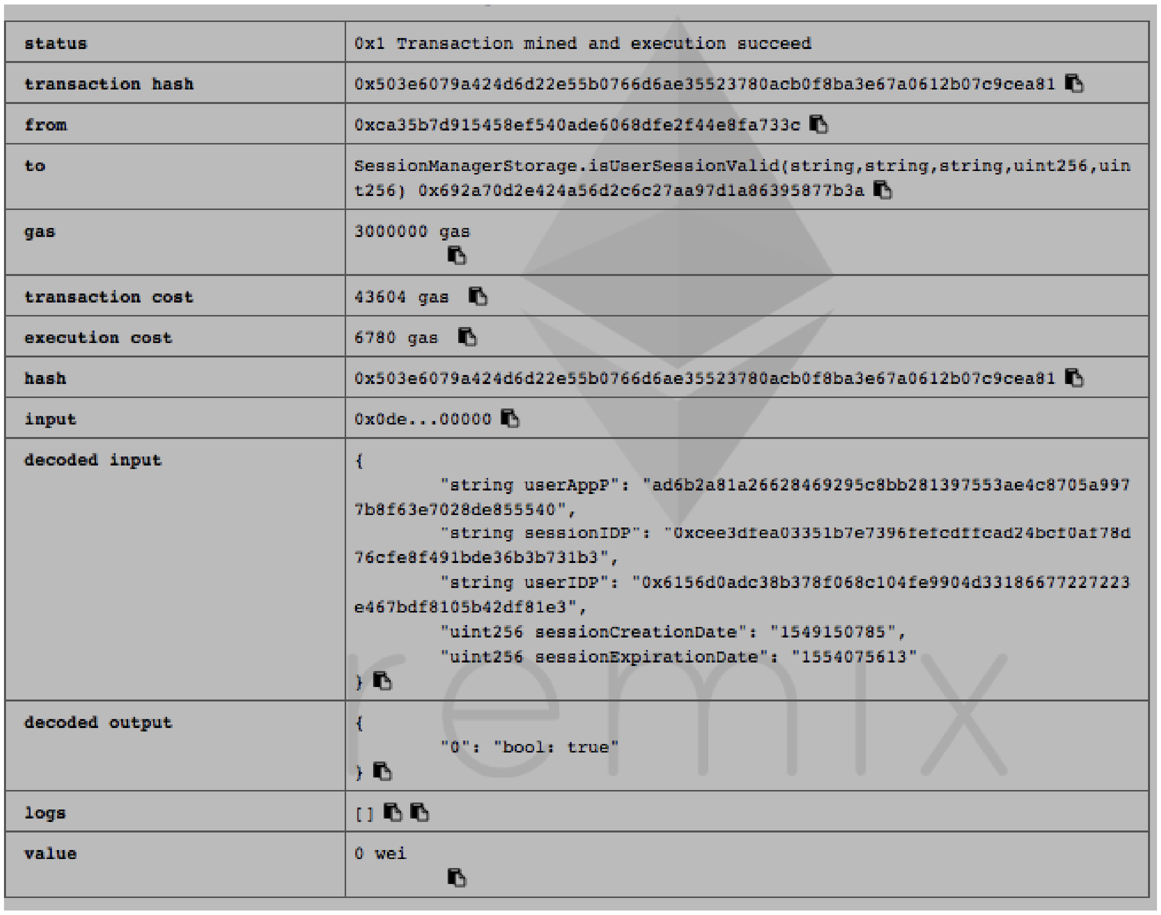


| | |
|---|---|
| status | 0x1 Transaction mined and execution succeed |
| transaction hash | 0x7e1758eeb1e55a2a954aaa2235a00c5ac13033ee62a5086c63510246c208aaf6 |
| from | 0xca35b7d915458ef540ade6068dfe2f44e8fa733c |
| to | SessionManagerStorage.createUserSession(string,string,string,uint256,uint256) 0x692a70d2e424a56d2c6c27aa97d1a86395877b3a |
| gas | 3000000 gas |
| transaction cost | 70576 gas |
| execution cost | 33752 gas |
| hash | 0x7e1758eeb1e55a2a954aaa2235a00c5ac13033ee62a5086c63510246c208aaf6 |
| input | 0x7a3...00000 |
| decoded input | { "string userAppP": "ad6b2a81a26628469295c8bb281397553ae4c8705a9977b8f63e7028de855540", "string sessionIDP": "0xcee3dfea03351b7e7396fefcdffcad24bcf0af78d76cfe8f491bde36b3b731b3", "string userIDP": "0x6156d0adc38b378f068c104fe9904d33186677227223e467bdf8105b42df81e3", "uint256 sessionCreationDate": "1549150785", "uint256 sessionExpirationDate": "1554075613" } |
| decoded output | { "0": "bool: true" } |
| logs | [] |
| value | 0 wei |

Figure 3.6: Transaction generated for invoking create user session

### Validate a user session - valid session

The user is sending a request, after login, to the web/enterprise application. The data is valid as well as the creation and expiration dates versus the current date-time. Result: The Blockchain validates the session, and it is returning ok. Fig. 3.7.

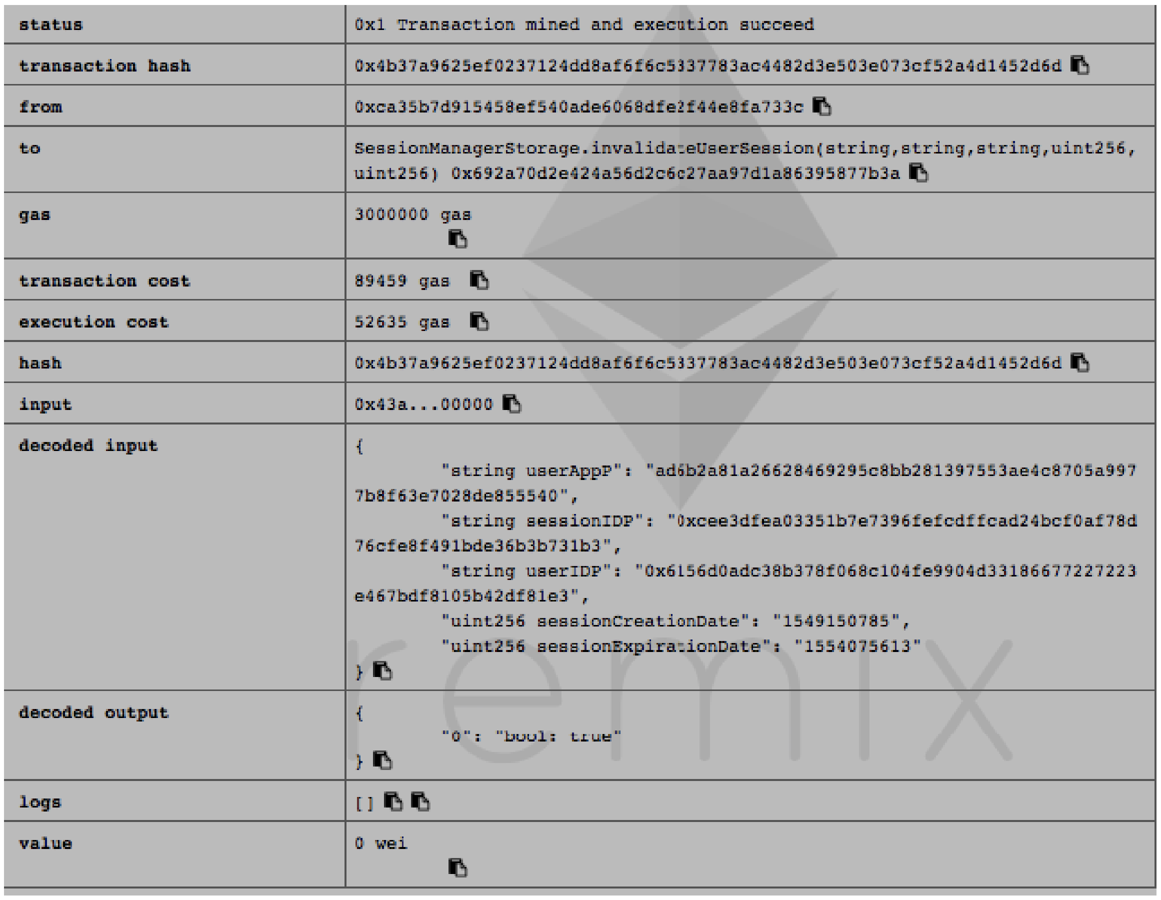


| | |
|---|---|
| status | 0x1 Transaction mined and execution succeed |
| transaction hash | 0x503e6079a424d6d22e55b0766d6ae35523780acb0f8ba3e67a0612b07c9cea81 |
| from | 0xca35b7d915458ef540ade6068dfe2f44e8fa733c |
| to | SessionManagerStorage.isUserSessionValid(string,string,string,uint256,uint256) 0x692a70d2e424a56d2c6c27aa97d1a86395877b3a |
| gas | 3000000 gas |
| transaction cost | 43604 gas |
| execution cost | 6780 gas |
| hash | 0x503e6079a424d6d22e55b0766d6ae35523780acb0f8ba3e67a0612b07c9cea81 |
| input | 0x0de...00000 |
| decoded input | { "string userAppP": "ad6b2a81a26628469295c8bb281397553ae4c8705a9977b8f63e7028de855540", "string sessionIDP": "0xcee3dfea03351b7e7396fefcdffcad24bcf0af78d76cfe8f491bde36b3b731b3", "string userIDP": "0x6156d0adc38b378f068c104fe9904d33186677227223e467bdf8105b42df81e3", "uint256 sessionCreationDate": "1549150785", "uint256 sessionExpirationDate": "1554075613" } |
| decoded output | { "0": "bool: true" } |
| logs | [] |
| value | 0 wei |

Figure 3.7: Transaction generated for invoking validate user session (valid session)

### Invalidate a user session

The user logged out from the web/enterprise application or the application decided to finish the user session due to expiration, or an idle session. Result: The session was set as expired. Fig. 3.8.



| | |
|---|---|
| status | 0x1 Transaction mined and execution succeed |
| transaction hash | 0x4b37a9625ef0237124dd8af6f6c5337783ac4482d3e503e073cf52a4d1452d6d |
| from | 0xca35b7d915458ef540ade6068dfe2f44e8fa733c |
| to | SessionManagerStorage.invalidateUserSession(string,string,string,uint256,uint256) 0x692a70d2e424a56d2c6c27aa97d1a86395877b3a |
| gas | 3000000 gas |
| transaction cost | 89459 gas |
| execution cost | 52635 gas |
| hash | 0x4b37a9625ef0237124dd8af6f6c5337783ac4482d3e503e073cf52a4d1452d6d |
| input | 0x43a...00000 |
| decoded input | { "string userAppP": "ad6b2a81a26628469295c8bb281397553ae4c8705a9977b8f63e7028de855540", "string sessionIDP": "0xcee3dfea03351b7e7396fefcdffcad24bcf0af78d76cfe8f491bde36b3b731b3", "string userIDP": "0x6156d0adc38b378f068c104fe9904d33186677227223e467bdf8105b42df81e3", "uint256 sessionCreationDate": "1549150785", "uint256 sessionExpirationDate": "1554075613" } |
| decoded output | { "0": "bool: true" } |
| logs | [] |
| value | 0 wei |

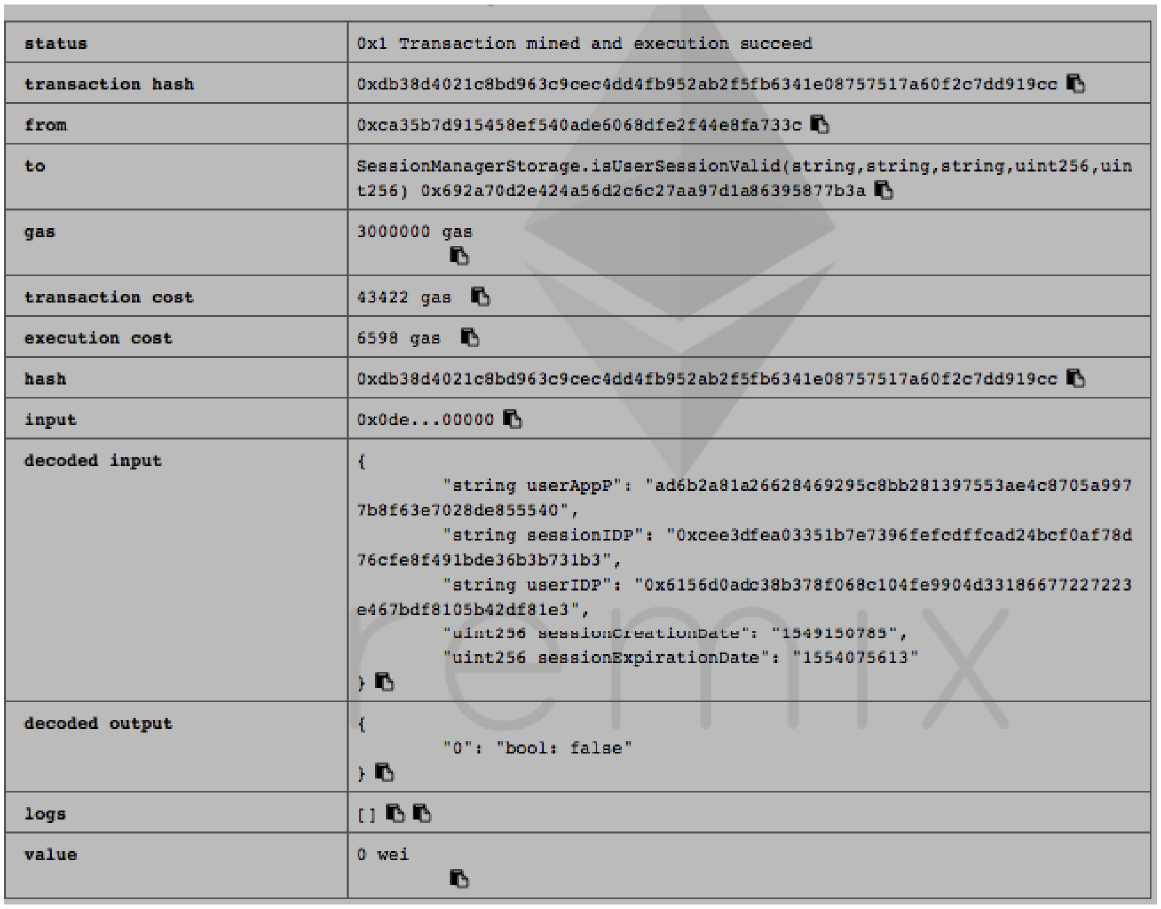
Figure 3.8: Transaction generated for invoking invalidate user session

**Validate a user session after invalidating the session - session expired**

The session was expired, and the user tries to access the application without login. Result: The user will not be able to access without previous login.

**Validate a user session after invalidating the session - end date expired**

The session was valid, but the end date was expired; the Blockchain will expire the session and returning false to the system. Result: The user will not be able to access without a previous login. Fig. 3.9.



| | |
|---|---|
| status | 0x1 Transaction mined and execution succeed |
| transaction hash | 0xdb38d4021c8bd963c9cec4dd4fb952ab2f5fb6341e08757517a60f2c7dd919cc |
| from | 0xca35b7d915458ef540ade6068dfe2f44e8fa733c |
| to | SessionManagerStorage.isUserSessionValid(string,string,string,uint256,uint256) 0x692a70d2e424a56d2c6c27aa97d1a86395877b3a |
| gas | 3000000 gas |
| transaction cost | 43422 gas |
| execution cost | 6598 gas |
| hash | 0xdb38d4021c8bd963c9cec4dd4fb952ab2f5fb6341e08757517a60f2c7dd919cc |
| input | 0x0de...00000 |
| decoded input | {<br>"string userAppP": "ad6b2a81a26628469295c8bb281397553ae4c8705a9977b8f63e7028de855540",<br>"string sessionIDP": "0xcee3dfea03351b7e7396fefcdffcad24bcf0af78d76cfe8f491bde36b3b731b3",<br>"string userIDP": "0x6156d0adc38b378f068c104fe9904d33186677227223e467bdf8105b42df81e3",<br>"uint256 sessionCreationDate": "1549150785",<br>"uint256 sessionExpirationDate": "1554075613"<br>} |
| decoded output | {<br>"0": "bool: false"<br>} |
| logs | [] |
| value | 0 wei |

Figure 3.9: Transaction generated for invoking validate user session (invalid session)

# Chapter 4

# Analysis

This analysis involves different scenarios for showing web and enterprise applications can be vulnerable to broken authentication. The output of this analysis could be different if custom approaches are implemented or software versions differ from these implementations. Also, this analysis is based on Drupal, Content Management system, which session management approach could differ from a custom Php Web application.

Why Drupal? Drupal is a Content Management system with more than a million users as part of this community [5].

Why Java? According to Oracle, in 2015, there were ”more than 12 million developers running Java and more than 15 billion installed devices” [12].

This research proposed a universal solution for different platforms; hence two main platforms were chosen to prove the hypothesis.

To prove the hypothesis different techniques has been executed, such as penetration testing and vulnerability scan.

## 4.1 Running vulnerabilities scan

The tool used was Burp Suite Professional which runs different scans to show if the application is vulnerable to any possible attacks. Each vulnerability includes the specific Common Weakness Enumeration (CWE) [6].

### 4.1.1 Php Web application- Drupal

Some of the Common Weakness Enumeration (CWE) found related to the authentication mechanism were:

#### Email Addresses disclosed

-Issue detail: The following email address was disclosed in the response: xxx@gmail.com

The user's email was included in the response without the need of being included.

-Vulnerability classification: CWE-200: Information Exposure

#### Password field with autocomplete enabled

-Issue detail: The page contains a form with the following action URL: http://localhost./drupal-8.3.7/user/login The form contains the following password field with autocomplete enabled: pass

Even though this functionality allows users to have a better user experience, this information is stored in the local computer and then could be accessed if the attacker gains access.

-Vulnerability classification: CWE-200: Information Exposure

#### Input returned in response (reflected)

-Issue detail: "The name of an arbitrarily supplied URL parameter is copied into the application's response."

Inputs returned in a response could help attackers to perform other attacks such as SQL injection.

-Vulnerability classifications: CWE-20: Improper Input Validation CWE-116: Improper Encoding or Escaping of Output

#### Session Token in URL

-Issue detail: ”The response contains the following links that appear to contain session tokens: http://localhost./drupal-8.3.7/devel/cache/clear?destination=/drupal-8.3.7/user/1token=kSsf-P8F$_FN7mwDfOl_hol9Bby13a6Q_H4KUacfS18http : //localhost./drupal{-}8.3.7/devel/run{-}cron?destination = /drupal - 8.3.7/user/1token = TOKENID$”.

Even though the URL is only use by administrators this could lead to a session fixation, user impersonation and also exposing the information in the URL makes the attacker’s job more easy.

-Vulnerability classifications: CWE-200: Information Exposure CWE-384: Session Fixation CWE-598: Information Exposure Through Query Strings in GET Request

### 4.1.2 Enterprise applications- Java Spring Framework

#### Password field with autocomplete enabled

-Issue detail: The page contains a form with the following action URL: http://localhost.:8080/login The form contains the following password field with autocomplete enabled: password

Even though this functionality allows users to have a better user experience, this information is stored in the local computer and then could be accessed if the attacker gains access. Also, it could be accessible to be exploited by XSS (Cross Site Scripting).

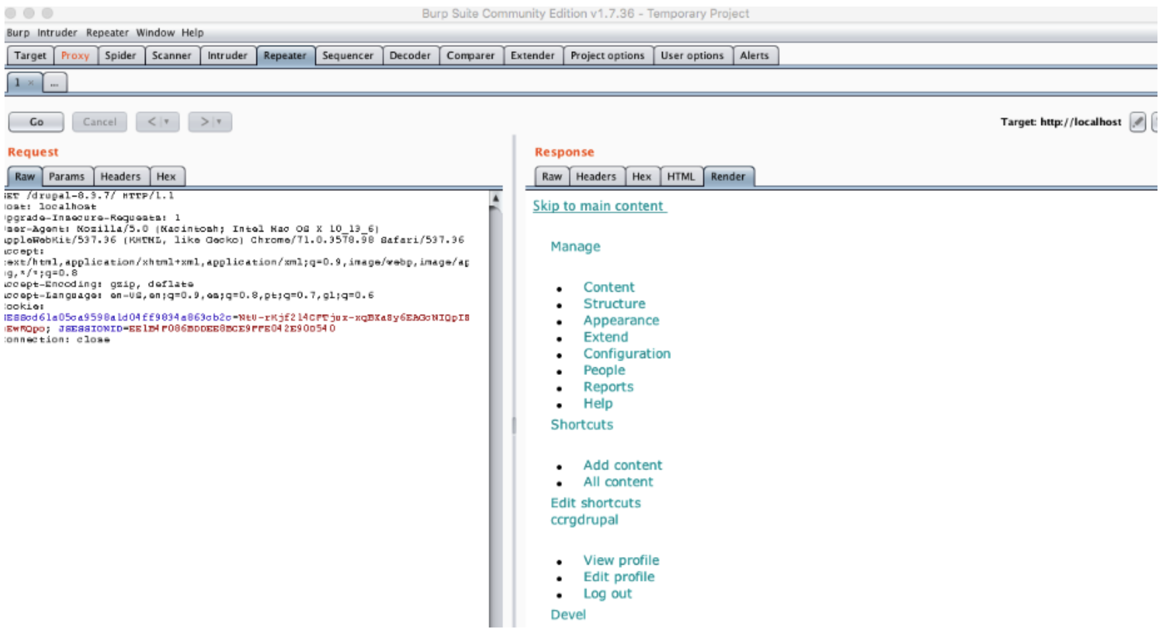
Vulnerability classification: CWE-200: Information Exposure

## 4.2 Penetration testing

### 4.2.1 Php Web applications

#### Login

Intercepting the request to login the user. I was able to login as a user using the user cookie information. Fig. 4.1.


Burp Suite Community Edition v1.7.36 - Temporary Project
Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Project options User options Alerts
Go Cancel
Target: http://localhost
Request
Raw Params Headers Hex
Response
Raw Headers Hex HTML Render

```
ET /drupal-8.3.7/ HTTP/1.1
ost: localhost
pgrade-Insecure-Requests: 1
ser-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_6)
ppleWebKit/537.36 (KHTML, like Gecko) Chrome/71.0.3578.98 Safari/537.36
ccept:
ext/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/ap
g,*/*;q=0.8
ccept-Encoding: gzip, deflate
ccept-Language: en-US,en;q=0.9,es;q=0.8,pt;q=0.7,gl;q=0.6
ookie:
ESSod61a05ca9598a1d04ff9834a863cb2c=NtU-rMjf214CFTjux-xqBXa8y6EAGcNIQpI8
EwRQpo; JSESSIONID=EE1B4F086BDDEE8BCE9FFE042E900540
onnection: close
```

Skip to main content

Manage

- Content
- Structure
- Appearance
- Extend
- Configuration
- People
- Reports
- Help

Shortcuts

- Add content
- All content

Edit shortcuts
ccrgdrupal

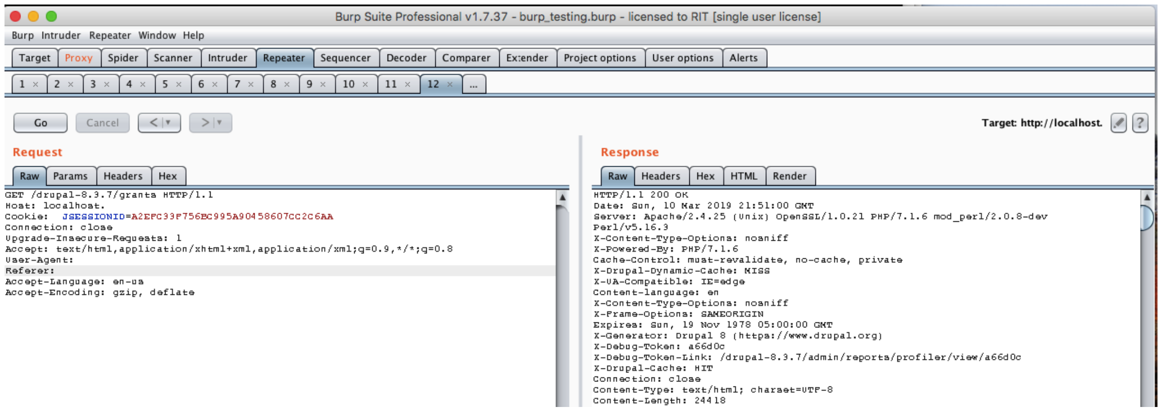
- View profile
- Edit profile
- Log out

Devel

Figure 4.1: Penetration testing with Burp - Drupal login

## Request

I was able to intercept the request made by a logged user and generate other requests by including the JSESSIONID, I removed other tokens, user agent information and the Referer and the application returned all the information anyway. Fig. 4.2.


Burp Suite Professional v1.7.37 - burp_testing.burp - licensed to RIT [single user license]
Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Project options User options Alerts
1 2 3 4 5 6 7 8 9 10 11 12 ...
Go Cancel
Target: http://localhost.
Request
Raw Params Headers Hex
Response
Raw Headers Hex HTML Render

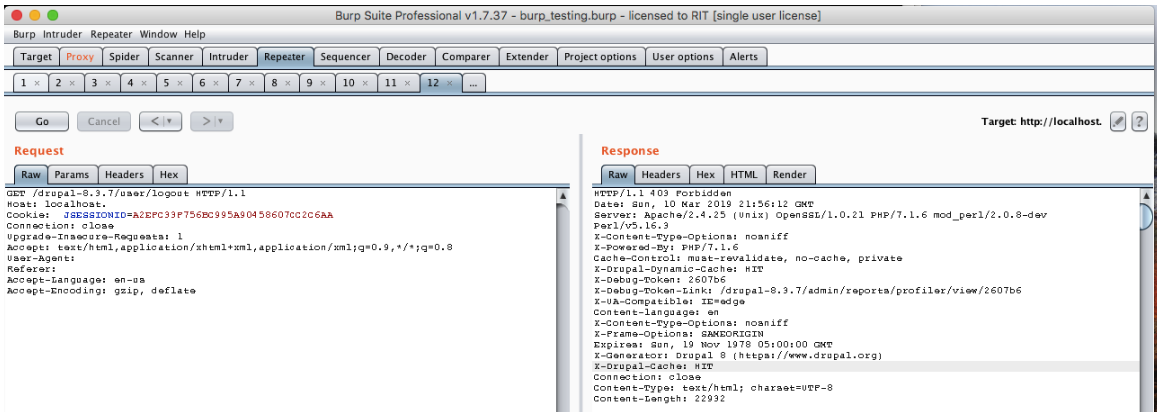
```
GET /drupal-8.3.7/grants HTTP/1.1
Host: localhost.
Cookie:  JSESSIONID=A2EFC33F756BC995A90458607CC2C6AA
Connection: close
Upgrade-Insecure-Requests: 1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
User-Agent:
Referer:
Accept-Language: en-us
Accept-Encoding: gzip, deflate
```

```
HTTP/1.1 200 OK
Date: Sun, 10 Mar 2019 21:51:00 GMT
Server: Apache/2.4.25 (Unix) OpenSSL/1.0.2l PHP/7.1.6 mod_perl/2.0.8-dev
Perl/v5.16.3
X-Content-Type-Options: nosniff
X-Powered-By: PHP/7.1.6
Cache-Control: must-revalidate, no-cache, private
X-Drupal-Dynamic-Cache: MISS
X-UA-Compatible: IE=edge
Content-language: en
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
Expires: Sun, 19 Nov 1978 05:00:00 GMT
X-Generator: Drupal 8 (https://www.drupal.org)
X-Debug-Token: a66d0c
X-Debug-Token-Link: /drupal-8.3.7/admin/reports/profiler/view/a66d0c
X-Drupal-Cache: HIT
Connection: close
Content-Type: text/html; charset=UTF-8
Content-Length: 24418
```

Figure 4.2: Penetration testing with Burp - Drupal request

### Logout

I was not able to logout the user with only sending the JSESSIONID, Fig. 4.3. However, including the additional token is allowing me to logout, Fig. 4.4.


Burp Suite Professional v1.7.37 - burp_testing.burp - licensed to RIT [single user license]
Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Project options User options Alerts
Go Cancel
Target: http://localhost.
Raw Params Headers Hex
Raw Headers Hex HTML Render

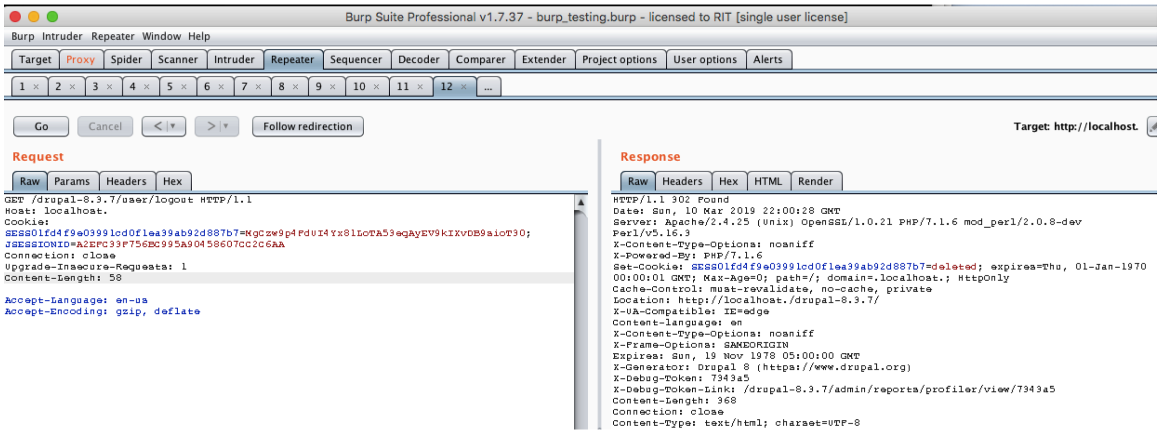
Request

```
GET /drupal-8.3.7/user/logout HTTP/1.1
Host: localhost.
Cookie:  JSESSIONID=A2EFC33F756BC995A90458607CC2C6AA
Connection: close
Upgrade-Insecure-Requests: 1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
User-Agent:
Referer:
Accept-Language: en-us
Accept-Encoding: gzip, deflate
```

Response

```
HTTP/1.1 403 Forbidden
Date: Sun, 10 Mar 2019 21:56:12 GMT
Server: Apache/2.4.25 (Unix) OpenSSL/1.0.2l PHP/7.1.6 mod_perl/2.0.8-dev
Perl/v5.16.3
X-Content-Type-Options: nosniff
X-Powered-By: PHP/7.1.6
Cache-Control: must-revalidate, no-cache, private
X-Drupal-Dynamic-Cache: HIT
X-Debug-Token: 2607b6
X-Debug-Token-Link: /drupal-8.3.7/admin/reports/profiler/view/2607b6
X-UA-Compatible: IE=edge
Content-language: en
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
Expires: Sun, 19 Nov 1978 05:00:00 GMT
X-Generator: Drupal 8 (https://www.drupal.org)
X-Drupal-Cache: HIT
Connection: close
Content-Type: text/html; charset=UTF-8
Content-Length: 22932
```

Figure 4.3: Penetration testing with Burp - Drupal logout


Burp Suite Professional v1.7.37 - burp_testing.burp - licensed to RIT [single user license]
Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Project options User options Alerts
Go Cancel Follow redirection
Target: http://localhost.
Raw Params Headers Hex
Raw Headers Hex HTML Render

Request

```
GET /drupal-8.3.7/user/logout HTTP/1.1
Host: localhost.
Cookie:
SESS01fd4f9e03991cd0f1ea39ab92d887b7=MgCzw9p4FdUI4Yx81LoTA53eqAyEV9kIXvDB9aioT30;
JSESSIONID=A2EFC33F756BC995A90458607CC2C6AA
Connection: close
Upgrade-Insecure-Requests: 1
Content-Length: 58

Accept-Language: en-us
Accept-Encoding: gzip, deflate
```

Response

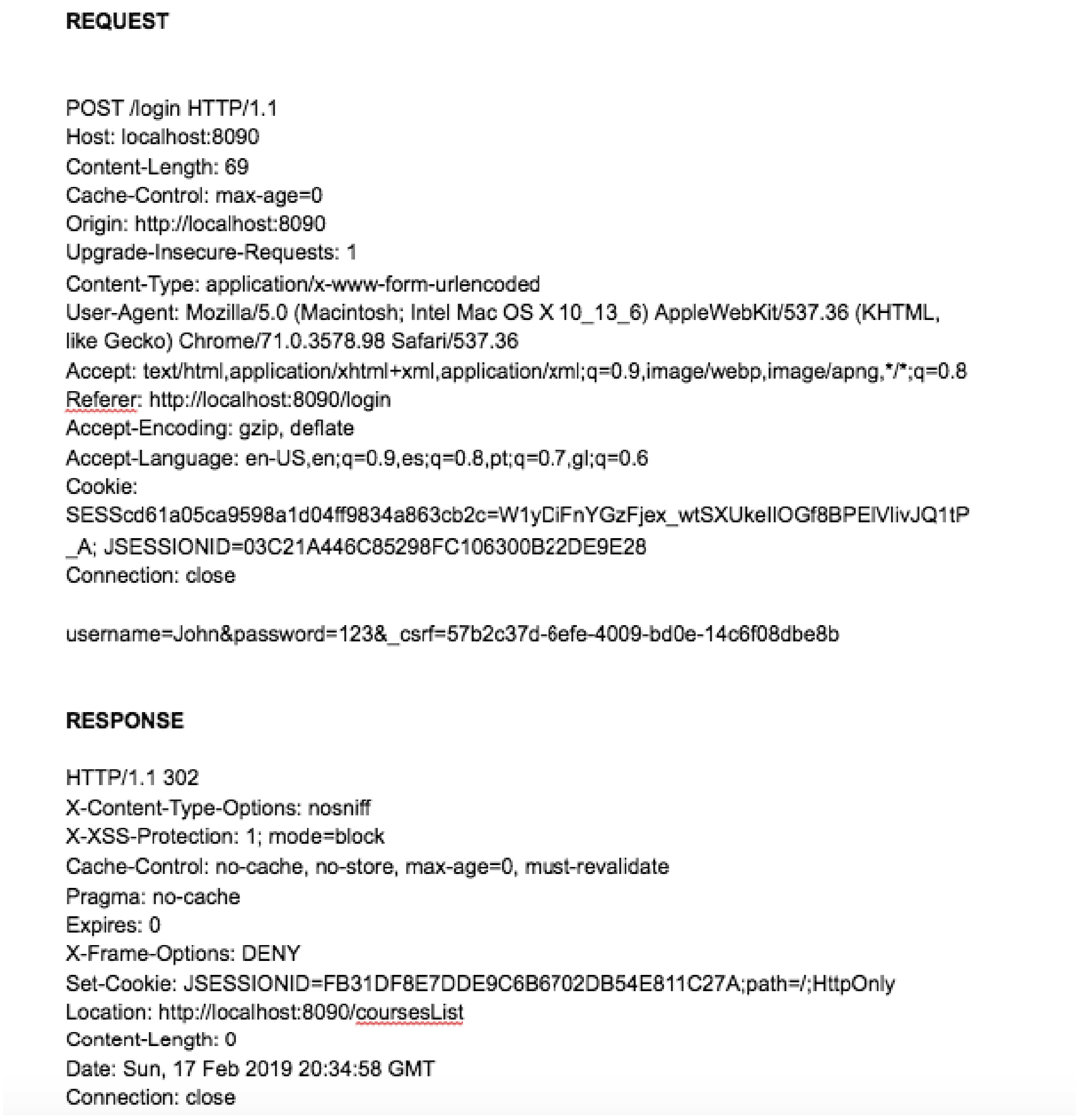
```
HTTP/1.1 302 Found
Date: Sun, 10 Mar 2019 22:00:28 GMT
Server: Apache/2.4.25 (Unix) OpenSSL/1.0.2l PHP/7.1.6 mod_perl/2.0.8-dev
Perl/v5.16.3
X-Content-Type-Options: nosniff
X-Powered-By: PHP/7.1.6
Set-Cookie: SESS01fd4f9e03991cd0f1ea39ab92d887b7=deleted; expires=Thu, 01-Jan-1970
00:00:01 GMT; Max-Age=0; path=/; domain=.localhost.; HttpOnly
Cache-Control: must-revalidate, no-cache, private
Location: http://localhost./drupal-8.3.7/
X-UA-Compatible: IE=edge
Content-language: en
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
Expires: Sun, 19 Nov 1978 05:00:00 GMT
X-Generator: Drupal 8 (https://www.drupal.org)
X-Debug-Token: 7343a5
X-Debug-Token-Link: /drupal-8.3.7/admin/reports/profiler/view/7343a5
Content-Length: 368
Connection: close
Content-Type: text/html; charset=UTF-8
```

Figure 4.4: Penetration testing with Burp - Drupal - a successful logout

## 4.2.2 Enterprise applications- Java Spring Framework

### Login

Intercepting the request to login the user. I was able to login as a user using the user cookie information. Fig. 4.5.

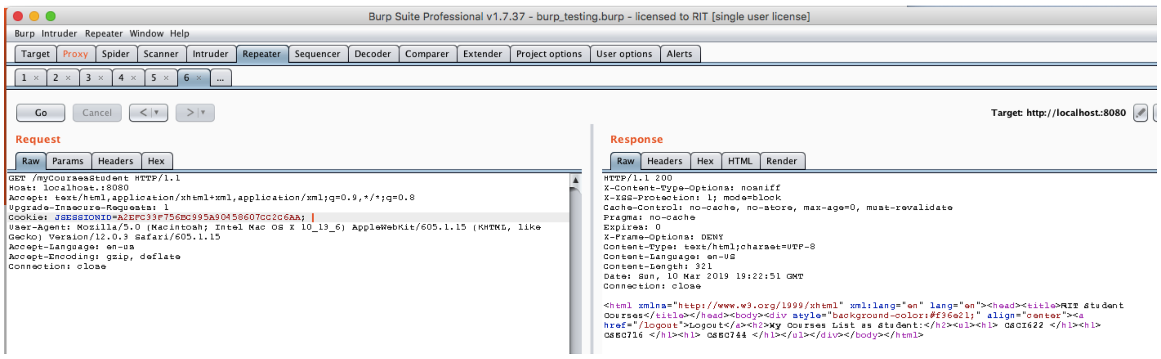


```
REQUEST

POST /login HTTP/1.1
Host: localhost:8090
Content-Length: 69
Cache-Control: max-age=0
Origin: http://localhost:8090
Upgrade-Insecure-Requests: 1
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_6) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/71.0.3578.98 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
Referer: http://localhost:8090/login
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9,es;q=0.8,pt;q=0.7,gl;q=0.6
Cookie:
SESScd61a05ca9598a1d04ff9834a863cb2c=W1yDiFnYGzFjex_wtSXUkellOGf8BPEIVlivJQ1tP
_A; JSESSIONID=03C21A446C85298FC106300B22DE9E28
Connection: close

username=John&password=123&_csrf=57b2c37d-6efe-4009-bd0e-14c6f08dbe8b

RESPONSE

HTTP/1.1 302
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Pragma: no-cache
Expires: 0
X-Frame-Options: DENY
Set-Cookie: JSESSIONID=FB31DF8E7DDE9C6B6702DB54E811C27A;path=/;HttpOnly
Location: http://localhost:8090/coursesList
Content-Length: 0
Date: Sun, 17 Feb 2019 20:34:58 GMT
Connection: close
```

Figure 4.5: Penetration testing with Burp - Java login

## Request

I was able to intercept the request made by a logged user and generate other requests by passing the user cookie session information. Fig. 4.6. Also, I removed other values, such as Cookie id, and it is allowing access by presenting only the JSESSIONID. If the attacker gain access to the JSESSIONID, he will be able to impersonate the user.


Burp Suite Professional v1.7.37 - burp_testing.burp - licensed to RIT [single user license]
Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Project options User options Alerts
1 2 3 4 5 6 ...
Go Cancel
Target: http://localhost.:8080
Request
Raw Params Headers Hex
Response
Raw Headers Hex HTML Render

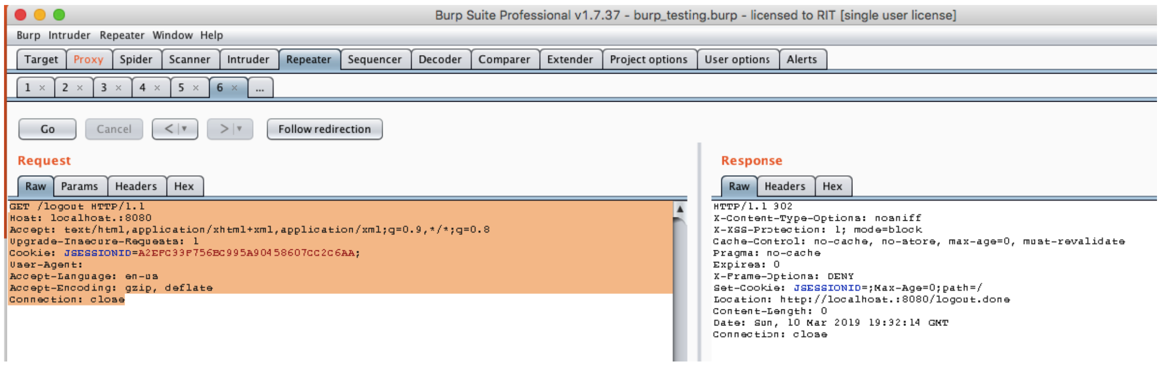
```
GET /myCoursesStudent HTTP/1.1
Host: localhost.:8080
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Upgrade-Insecure-Requests: 1
Cookie: JSESSIONID=A2EFC33F756BC995A90458607CC2C6AA;
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_6) AppleWebKit/605.1.15 (KHTML, like
Gecko) Version/12.0.3 Safari/605.1.15
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: close
```

```
HTTP/1.1 200
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Pragma: no-cache
Expires: 0
X-Frame-Options: DENY
Content-Type: text/html;charset=UTF-8
Content-Language: en-US
Content-Length: 321
Date: Sun, 10 Mar 2019 19:22:51 GMT
Connection: close

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en"><head><title>RIT Student
Courses</title></head><body><div style="background-color:#f36e21;" align="center"><a
href="/logout">Logout</a><h2>My Courses List as Student:</h2><ul><h1> CSCI622 </h1><h1>
CSEC716 </h1><h1> CSEC744 </h1></ul></div></body></html>
```

Figure 4.6: Penetration testing with Burp - Java request

## Logout

I was able to logout as an authenticated user with using only the JSESSIONID. I removed the Referer and User-Agent and it worked well without the need of other fields Fig. 4.7.


Burp Suite Professional v1.7.37 - burp_testing.burp - licensed to RIT [single user license]
Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Project options User options Alerts
1 2 3 4 5 6 ...
Go Cancel Follow redirection
Request
Raw Params Headers Hex
Response
Raw Headers Hex

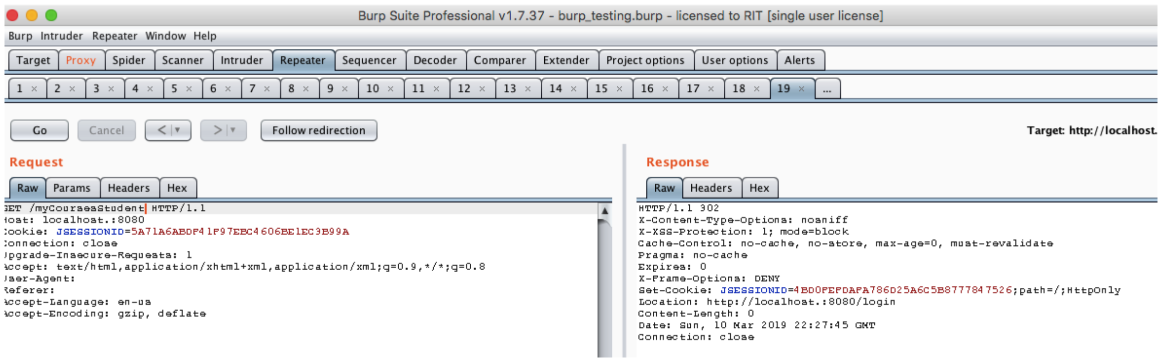
```
GET /logout HTTP/1.1
Host: localhost.:8080
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Upgrade-Insecure-Requests: 1
Cookie: JSESSIONID=A2EFC33F756BC995A90458607CC2C6AA;
User-Agent:
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: close
```

```
HTTP/1.1 302
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Pragma: no-cache
Expires: 0
X-Frame-Options: DENY
Set-Cookie: JSESSIONID=;Max-Age=0;path=/
Location: http://localhost.:8080/logout.done
Content-Length: 0
Date: Sun, 10 Mar 2019 19:32:14 GMT
Connection: close
```

Figure 4.7: Penetration testing with Burp - Java logout

## After Logout

I was not able to access other pages; the application redirected me to the login page, Fig. 4.8.


Burp Suite Professional v1.7.37 - burp_testing.burp - licensed to RIT [single user license]
Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Project options User options Alerts
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 ...
Go Cancel Follow redirection
Target: http://localhost.
Request
Raw Params Headers Hex
Response
Raw Headers Hex

```
ET /myCoursesStudent HTTP/1.1
ost: localhost.:8080
ookie: JSESSIONID=5A71A6ABDF41F97EBC4606BE1EC3B99A
onnection: close
pgrade-Insecure-Requests: 1
ccept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
ser-Agent:
eferer:
ccept-Language: en-us
ccept-Encoding: gzip, deflate
```

```
HTTP/1.1 302
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Pragma: no-cache
Expires: 0
X-Frame-Options: DENY
Set-Cookie: JSESSIONID=4BD0FEFDAFA786D25A6C5B87778 47526;path=/;HttpOnly
Location: http://localhost.:8080/login
Content-Length: 0
Date: Sun, 10 Mar 2019 22:27:45 GMT
Connection: close
```

Figure 4.8: Penetration testing with Burp - Java after logout

# Chapter 5

# Discussion and Conclusion

## 5.1 Conclusions based on penetration testing

In both applications, I was able to access as an authenticated user impersonating the user logged in, by only using its JSESSIONID. I removed cookie's information as well as user agent and I was able to access different scenarios inside the application.

Java Enterprise implementation did not show any high vulnerability in the scan, as soon as an attacker gains access to the JSESSIONID, he will be accessing as an authenticated user. However, for Drupal several vulnerabilities were found and one of those was exposing the SESSIONID.

How easy is to predict the JSESSIONID? According to Burp Sequencer, the ID generated is not random, which makes the Session ID more difficult to predict. JSESSIONID could be gain by the attacker using XSS,MITM, MITB.

Java Enterprise implementation validates that the token is still valid (ie. the user did not logout). So, after logout I was not able to access again with the same token.

Therefore, I have shown that session Ids are vulnerable and are a cornerstone in the authentication mechanism for stateless communication. The session ID can be predicted by another system or can be stolen or shared with other users. Hence, the session ID needs additional constraints besides the session ID complexity. The approach presented in this research proposed robust authentication mechanisms that does not rely only on the application but instead creates a relationship between a user and its session data. Therefore, this work showed that it is possible to exploit the characteristics of Blockchain to prevent session management attacks from being successful.

## 5.2 Conclusions based on the smart contract implementation

Session management security is stronger since there is no possibility to login in an application without sending all valid information (session ID, user ID, expiration date, creation date). Also, restrictions about time and expiration have been applied. Once the session has expired, the session will not be longer available. A user cannot have more than one session available for the same application. These reduce the session ID lifetime, making more difficult to access and impersonate the user. Another aspect is the relationship created between the user and the session. Even when a hacker is able to see the data in the network, he will not be able to impersonate the user since the message will be associated with the user and his digital wallet. However, if the attacker gains access to the user's computer, the digital wallet could be comprised.

## 5.3 Current Status

The current implementation supports Ethereum Blockchain and was tested in a testing environment. Since, applications are not deployed in production, traffic and other constraints are not measured.

## 5.4 Future Work

Some improvements that could be addressed in future development are:

- Enable the application proposed in other Blockchain networks. Developing a multi-platform application will be beneficial for those who are currently supporting other Blockchain networks, such as Hyper-ledger.

- Consider storing additional data. Blockchain could be more flexible in the future. Currently, additional attributes cannot be added without changing the smart contract. However, this improvement could be create other issues, for example, we do not want to store private data in the Blockchain.

- Consider time constraints in this research, I did not measure time or consider it for this implementation. However, there is a need for collecting user time-response data. One proposal could include an internal private Blockchain implementation, thus reducing the time for transactions to be approved. However, this approach will exchange the decentralization mechanism for a centralized mechanism.

- Consider the need for ”money” or ”gas”. Ethereum needs ”gas” to deploy the smart contract and execute it. This monetary requirement could be an issue for small companies, and an optimal balance between resources. Some companies are resolving this problem by offering advertisements to providers.

## 5.5 Lessons Learned

### 5.5.1 Development environment for the smart contract

I spent too much time setting up the right environment for development. I faced many issues with scripts after realizing there are other tools for testing the smart contract. Once I started using Remix, it was much smoother and precise. The main problem was to make the code compile after using an IDE without Solidity support. The errors returned by the compiler were not apparent and not all the issues have a unique solution or a standard solution. There is not much documentation online for resolving compilation issues. The second problem was the testing, and some scripts did not work correctly, Remix worked very well. The debugging tool for Remix is very good, and I did not have any trouble.

### 5.5.2 Integration environment issues

Understanding different tools and environments was a challenge. First, I needed to understand which was the right tool for what I needed to do. Then, I had to try different tools and learn how to use them. I faced issues with some tools, and I could not find a solution on their websites. After I gave it some thought I realized that some of the solutions were not right for my system. This lack of documentation and solutions to tool problems hindered the integration effort in this implementation. This generated a time constraint in my research for integrating the whole solution.

# Bibliography

[1] Alessandro Armando, Roberto Carbone, Luca Compagna, Jorge Cuellar, and Llanos Tobarra. Formal analysis of saml 2.0 web browser single sign-on: breaking the saml-based single sign-on for google apps. In *Proceedings of the 6th ACM workshop on Formal methods in security engineering*, pages 1–10. ACM, 2008.

[2] Blockstack. Blockstack: Dthe easiest way to start building decentralized blockchain apps. https://blockstack.org/, Accessed October 1, 2017.

[3] John Bradley, Nat Sakimura, and Michael Jones. Json web token (jwt). 2015.

[4] Italo Dacosta, Saurabh Chakradeo, Mustaque Ahamad, and Patrick Traynor. One-time cookies: Preventing session hijacking attacks with stateless authentication tokens. *ACM Trans. Internet Technol.*, 12(1):1:1–1:24, July 2012.

[5] Drupal. Why drupal. https://www.drupal.org/about, Accessed March , 2019.

[6] Common Weakness Enumeration. the mitre corporation, 2018.

[7] Dick Hardt. The oauth 2.0 authorization framework. Technical report, 2012.

[8] Ori Jacobovitz. Blockchain for identity management. *The Lynne and William Frankel Center for Computer Science Department of Computer Science. Ben-Gurion University, Beer Sheva Google Scholar*, 2016.

[9] Mitja Kolšek. Session fixation vulnerability in web-based applications. *Acros Security*, 7, 2002.

[10] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system., 2008. https://bitcoin.org/bitcoin.pdf, Accessed July 20, 2017.

[11] onelogin. Jwt. https://jwt.io/introduction/, Accessed October 1, 2018.

[12] Oracle. 20 years of java. http://oracle.com.edgesuite.net/timeline/java/, Accessed March , 2019.

[13] OWASP. Owasp top ten project, 2019.

[14] Aaron Parecki. User authentication with oauth 2.0.

[15] Nat Sakimura, D Bradley, B de Mederiso, M Jones, and Edmund Jay. Openid connect standard 1.0-draft 07, 2011.

[16] Dafydd Stuttard and Marcus Pinto. *The web application hacker's handbook: Finding and exploiting security flaws*. John Wiley & Sons, 2011.

[17] The New York Times. Facebook security breach exposes accounts of 50 million users., October 2018.

[18] Ingo Weber, Vincent Gramoli, Alex Ponomarev, Mark Staples, Ralph Holz, An Binh Tran, and Paul Rimba. On availability for blockchain-based systems. In *Reliable Distributed Systems (SRDS), 2017 IEEE 36th Symposium on*, pages 64–73. IEEE, 2017.

[19] Shellie Wedman, Annette Tetmeyer, and Hossein Saiedian. An analytical study of web application session management mechanisms and http session hijacking attacks. *Information Security Journal: A Global Perspective*, 22(2):55–67, 2013.