# Implementation and verification of webinos authentication protocols

Report for the Computer Security exam at the Politecnico di Torino

Teodoro Montanaro (188924)
tutor: Andrea Atzeni

September 2013

---

## Contents

# 1 Introduction

The aim of this work is to develop unambiguous methods to verify algorithmically that webinos authentication methods do not contain deadlocks or similar critical states that can cause system crashes.

One of the major problems encountered developing software, especially when the software is made up of a lot of components, is to verify that it does exactly what we expect from it. For this reason, the best way to verify the correctness of each module that makes up the software is to check out each part as a stand-alone component, and it is exactly what we did for webinos.

Webinos is a project that wants to enable web applications and services to be used and shared consistently and securely over a broad spectrum of converged and connected devices, including mobile, PC, home media (TV) and in-car units.
It is made up of a lot of different modules and, more specifically, it plans to introduce some novel authentication methods, which should at the same time introduce user-friendly SSO (Single Sign-On) and preserve user privacy. So, this homework have the task to analyze algorithmically only the behaviour of the authentication modules and methods from a security point of view.

This purpose can be achieved using model checking, which is just a method created to let developers analyze algorithmically their own software. The verification is achieved by the definition of precise mathematical languages representing the system, its states, its transitions and its properties that can be verified by the model checker: a software that is able to check whether the properties are satisfied in a logic point of view.

# 2 Webinos

## 2.1 What is webinos

Webinos is an EU-funded project and affiliate program that have the aim to define and develop an Open Source Platform for the Future Internet (in the form of web runtime extensions), to enable web applications and services to be used and shared consistently and securely over a broad spectrum of converged and connected devices, including mobile, PC, home media (TV) and in-car units.

## 2.2 Webinos purposes and applications

The webinos project intends to provide a platform to allow developers to only have to create their own applications once, but to distribute them on every operating system and on every device. Moreover, it intends to let devices find each other and discover and use the services each of them offer.

Using webinos will bring advantages both for users and developers: the users will have the possibility to know exactly how their information will be treated and everyone will be able to test the accuracy, efficiency and efficacy of algorithms used, for example, for security. On the other hand, the developers will have the opportunity to focus their efforts only on the development of ideas and not on all the other aspects that have to be considered now (like security, connection to other devices, and so on ... ).

## 2.3 General Architecture description

The webinos architecture is centered around the concept of Personal Zone, which is a way to organise personal devices and services and that provides a basis for managing the user's devices, together with the services running on them. The Personal zone supports:

- Single Sign-On: Through authentication on a device and/or on an application, the user will be authenticated on the Personal Zone too. This avoids the need for establishing direct peering relationships between each pair of devices and allows stronger authentication with the services the user uses. Obviously the architecture also allows situations in which the user is offline.
- Shared model of the context, which enables applications to dynamically adapt to changes, and to increase usability exploiting the context.
- Synchronisation across the devices in the zone: this includes support for distributed authentication, as well as personal preferences, and replication of service-specific data, e.g. social contacts, and appointments. Synchronisation is essential for supporting offline usage.
- Discovery and access to services: The high level discovery API allows Web developers to search for all local services, or to filter by service type and context, or even to locate a named service instance. Remote discovery is based on existing user names and Email addresses, resolving to a URI for a Personal Zone.

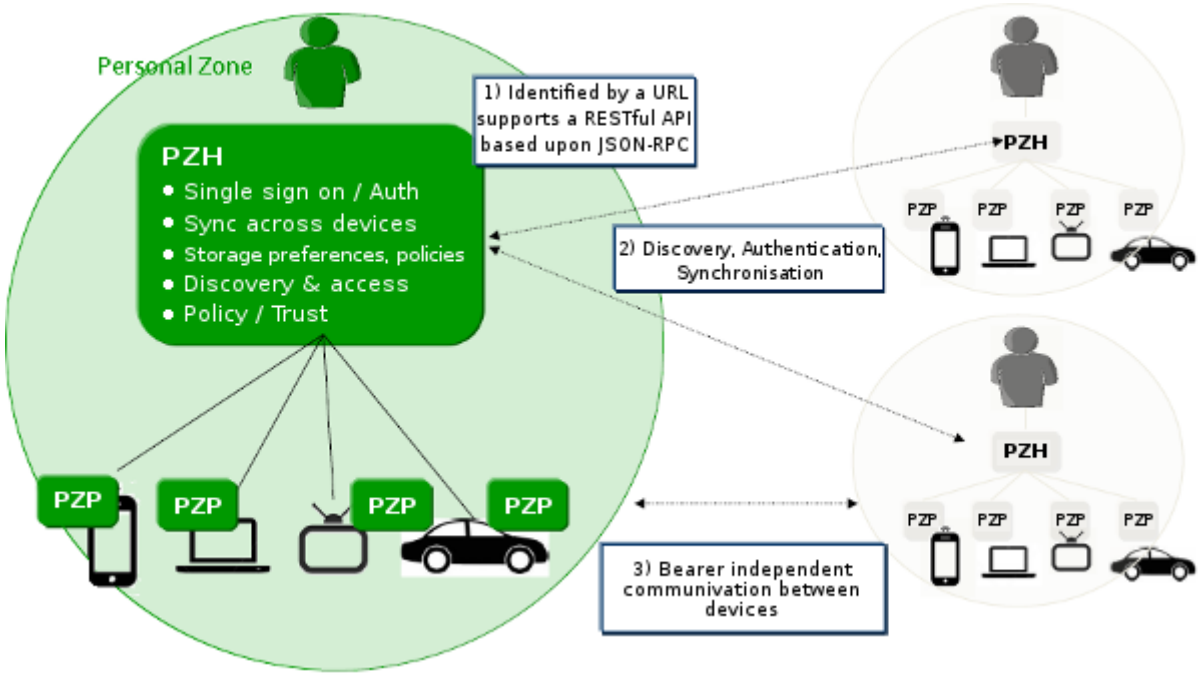


Figure 1: Personal Zones - reprinted from [4]

As shown in Fig. 1, a Personal Zone is implemented on a distributed basis, consisting of a single Personal Zone Hub (PZH) and multiple Personal Zone Proxies (PZPs).

There is a Personal Zone Hub for each user, which is identified by a URL and is associated with a one-to-one correlation to a Personal Zone. It is a service accessible via the Internet that contains all the information and data associated to the user.

This service is always accessible and operative so the user can use it by any device.

Moreover, a policy associated to every information is essential: everyone can specify what the other users can see about him/her.

On the device we need a software that interacts with the PZH and it is the Personal Zone Proxy.

There is a PZP on each device used by the user and it is able to work even if there is not an Internet connection.

For this reason, the PZP is responsible for the synchronization of all the data and information to let user use services even when the connection is not available.

All the services are accessible through a simple browser.

## 2.4 Authentication in webinos

### 2.4.1 How does it work?

There are three types of authentication available in webinos: device authentication, user authentication and authentication with third party services (for example Twitter).

**2.4.1.1 User authentication** Users are primarily authenticated through their OpenID credentials. This involves them connecting to their PZH web interface and logging in with their own account.

This first authentication is a pre-requisite for many sensitive operations, but is not sufficient for accessing webinos APIs or other device features. It only allows a user to connect through the PZH web interface. Applications do not have access to any enhanced privileges through this.

Alternatively, on some devices users may log into the PZH web interface using just client-side X509 certificates. This is possible only when that device has been configured within the personal zone (based on a configuration file at the personal zone hub) to support this.

As we can see in Fig. 2 (showing the sequence diagram of the method described above) there are 3 main system components: one that contains the User Agent and the javascript library, one that contains the PZH server and the OpenID Module and another one that contains the Identity Provider.

Let us analyze the sequence diagram, looking for the point in which it is important to guarantee high level security. The component containing the user agent is essentially a web browser in which the user opens the login page and presses the "Login" button. As result of this action an HTML page is loaded with its linked scripts and one of them is the *webinos javascript library*, which allows the web browser to load the authentication function contained in it. After that, this function sends a message to the *PZH webserver* saying that there is a login request. This is the first point in which we have to analyze if the security is at the right level: there is communication between 2 different components that could be located in different locations.

Due to what is reported in the documentation, in this method (*User authentication*) security is guaranteed by an *OpenID Provider Authentication Policy Extension over HTTPS*, which can provide:

- Authentication (simple / mutual): depending on the security methods available on the server and on the device, they can ask each other to demonstrate that they are exactly who they say they are.

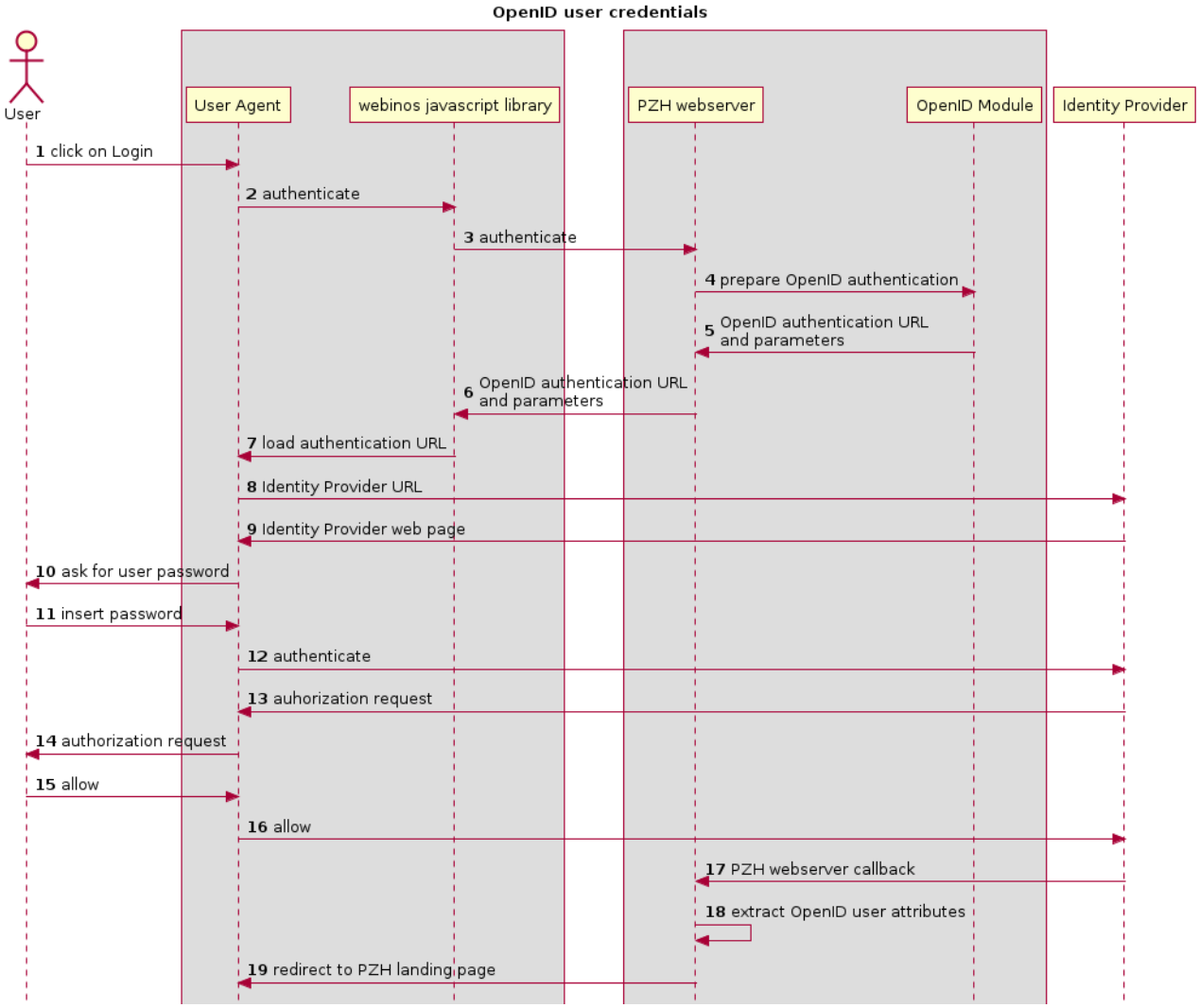


Figure 2: Sequence diagram relative to OpenID Authentication - reprinted from [4]

- Integrity: if someone intercepts their messages and modifies them, the use of keyed digest and message identifiers guarantees that the system can detect these modifications.
- Confidentiality: The system cyphers all the data sent by each part of the communication.

For this reason the only feasible attack in the list considered (2.4.2) is the DoS one (as shown in the results provided by the model checker 5.2).

Then, the *PZH webserver*, talking to the *OpenID module*, prepares the OpenID authentication parameters and URL and sends them to the *javascript library*. This is the second point in which we have to check for security.

After that the web browser contacts the *identity provider* (and here it is again important to test the efficiency of the security: the considerations are the same as discussed for the last point) and after some interactions with the user, the system will authenticate him on the *PZH server*.

As it is easy to understand looking at the above explanation of the diagram, security analysis is important in every case in which communication is established between 2 or more actors that are not in the same subsystem; the reason is that an intruder could modify or intercept any kind of message exchanged between them and could cause system crashes and/or program deadlocks.

**2.4.1.2 Device authentication** Devices are authenticated through the possession and use of an RSA private key. Having this key is a pre-requisite for a PZP connecting to a personal zone and sharing data with other devices, since it can be used to sign in and to prove device identity.

In Fig. 3 we can see the sequence diagram describing how a device can hold a private key.

As we can see, in this situation all the interactions and messages are exchanged within the same subsystem, so the only security problems could come from inside, but they are avoided by the use of the RSA private key (in addition to the use of HTTPS protocol, which is instead used to communicate with the PZH server at the end of the procedure) that lets the user cypher all the data.

Moreover, in order to make sure that a private key is used appropriately it must be integrated with the policy system and with the local device authentication. For instance, the private key will be held in a key storage, which requires the user to prove their presence unlocking it. Furthermore, it can interact with the authentication API, such that an application can work out whether the user is still present or not.

**2.4.1.3 Presence** There are different ways of interaction between devices and users: sometimes presence is assumed (smartphones) and sometimes it is not (shared TVs, home servers).

- Devices with assumed presence are defined as “private” devices. Their keys are marked as such. Use of these device keys implies that the user is present.
  - If a policy refers to just the end user, and not a specific device, then these devices will automatically match this policy. E.g. “Peter can access file X” would allow Peter to access file X from his “private” device.
- Devices without assumed presence (and this is the default) are defined as “shared” devices. Use of a shared device does not imply that the user is necessarily present.

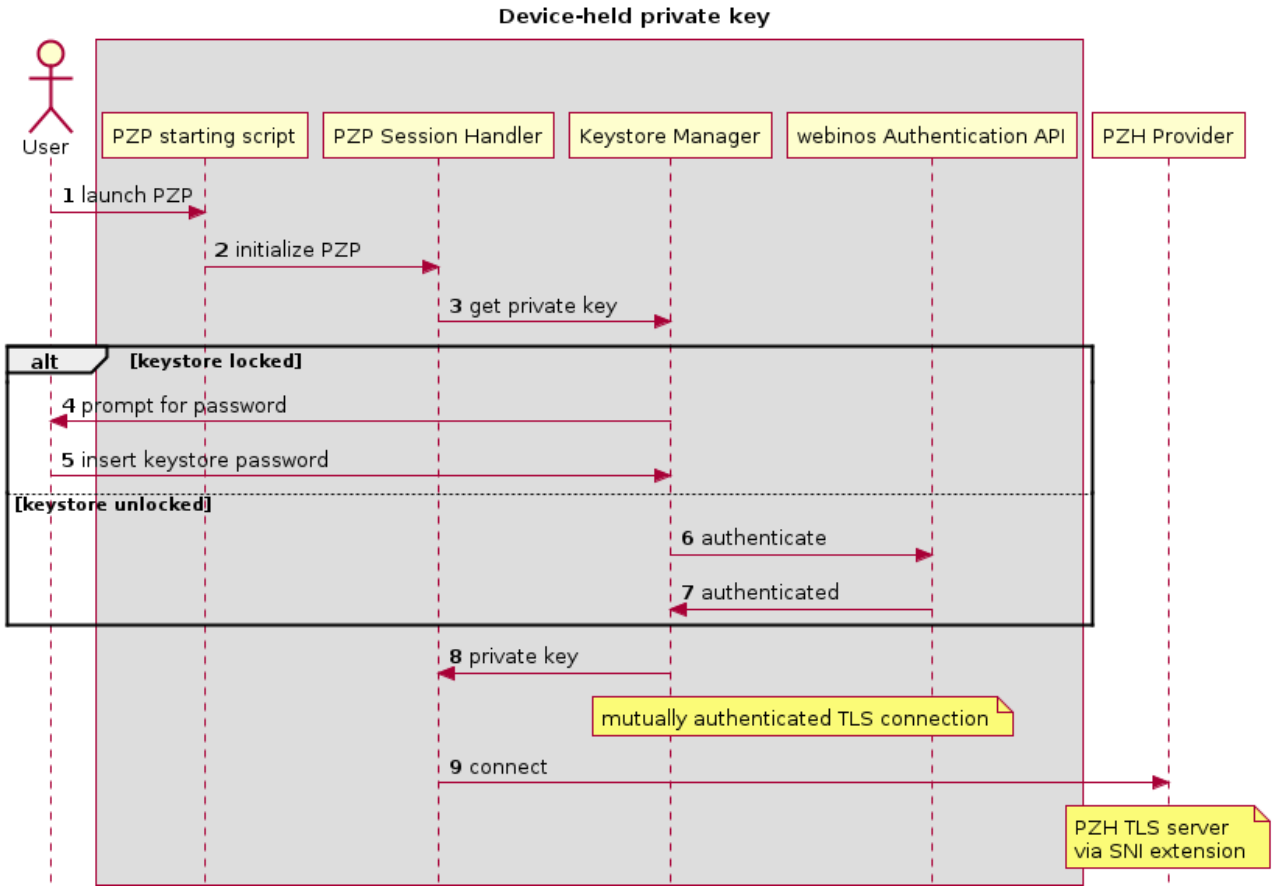


Figure 3: Sequence diagram relative to private key acquisition - reprinted from [4]

- If a policy refers to just the end user, and is about access to a local device API, then the user must re-authenticate using the Authentication API or OpenID.
- If a policy refers to just the end user, and is about access to a remote API, the user must re-authenticate using OpenID.

For these reasons every time a user wants to access resources the system has to check the user's presence, and it works as illustrated in Fig. 4: depending on the kind of assumed presence the system interacts with different entities that in any case ask the user to demonstrate that he is present (typing in his credentials).

In this situation security is managed using RSA private keys over HTTPS, so all the security properties shown at the previous point (Authentication, Integrity and Confidentiality) are satisfied and in addition the non repudiation property is guaranteed, because using RSA private keys we can certificate who the owner of every message/data sent is.

Figure 5 and Figure 6 show what the system does when a user attempts to perform a privileged action on the PZP: After the request the *PZP* forwards this request to the *PZH*, which replies requiring authentication. The message contains the URL for this authentication. Then, the *PZP* opens a web browser and directs the user to the URL provided. He must log into this URL and then the page presents an authorisation prompt explaining the action requested and the impact of the action on the personal zone. The user approves or rejects the prompt and the action is carried out or not, depending on the decision made.

In this situation the security properties (Authentication, Integrity, Confidentiality and non Repudiation) are guaranteed using RSA private key over HTTPS.

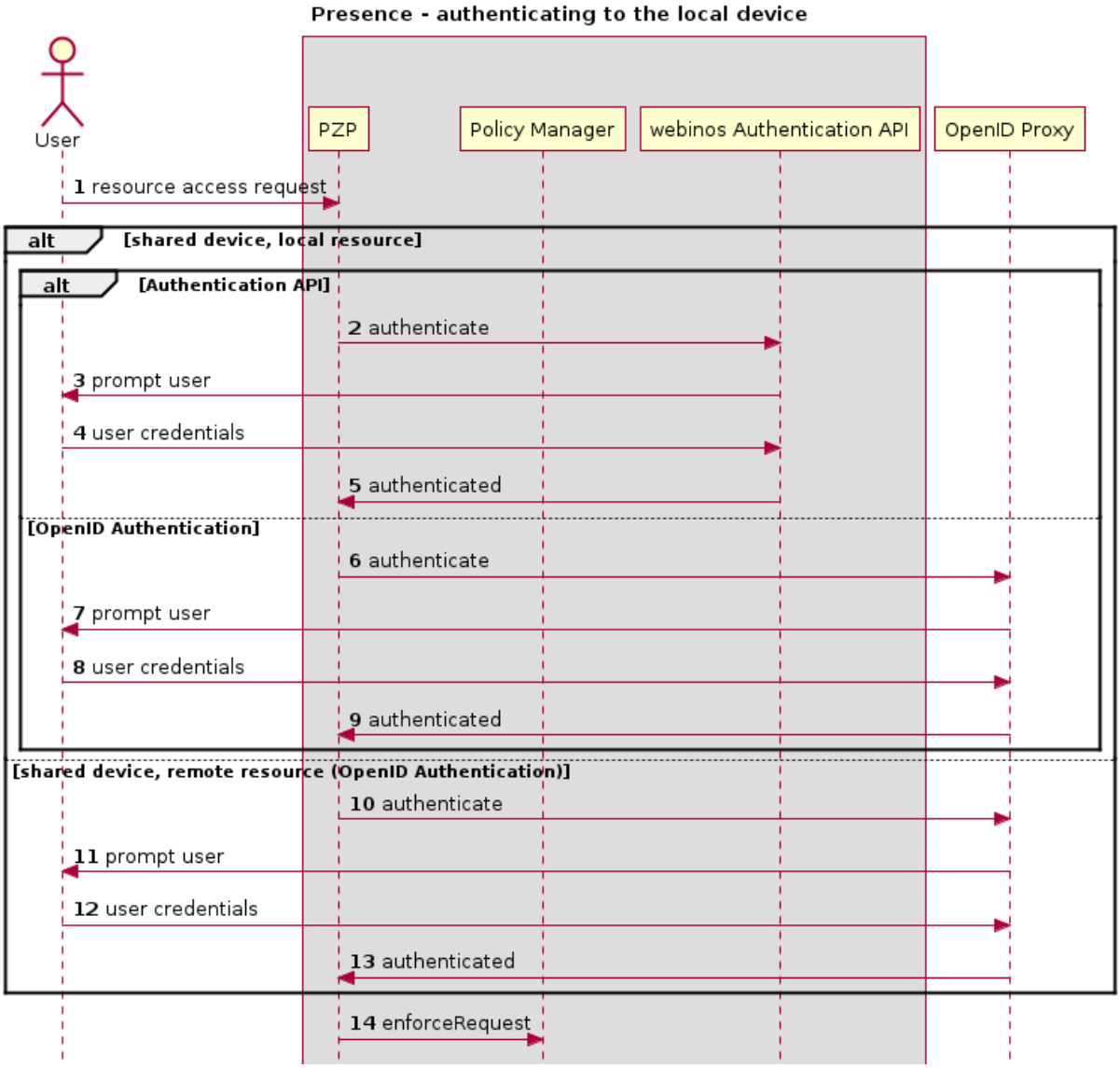


Figure 4: Presence: authenticating to the local device - reprinted from [4]

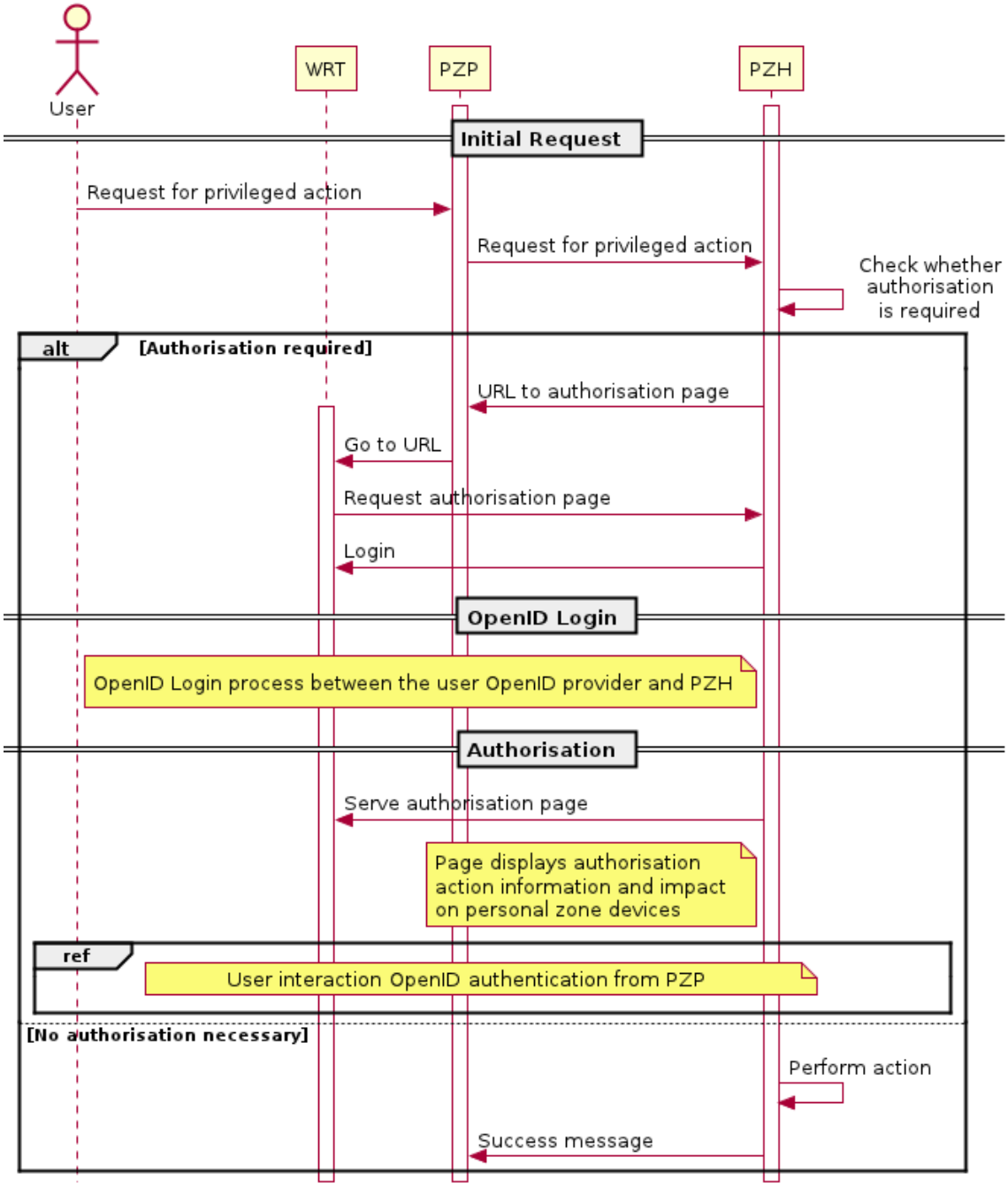


Figure 5: OpenID authentication from the PZP part n.1 - reprinted from [4]

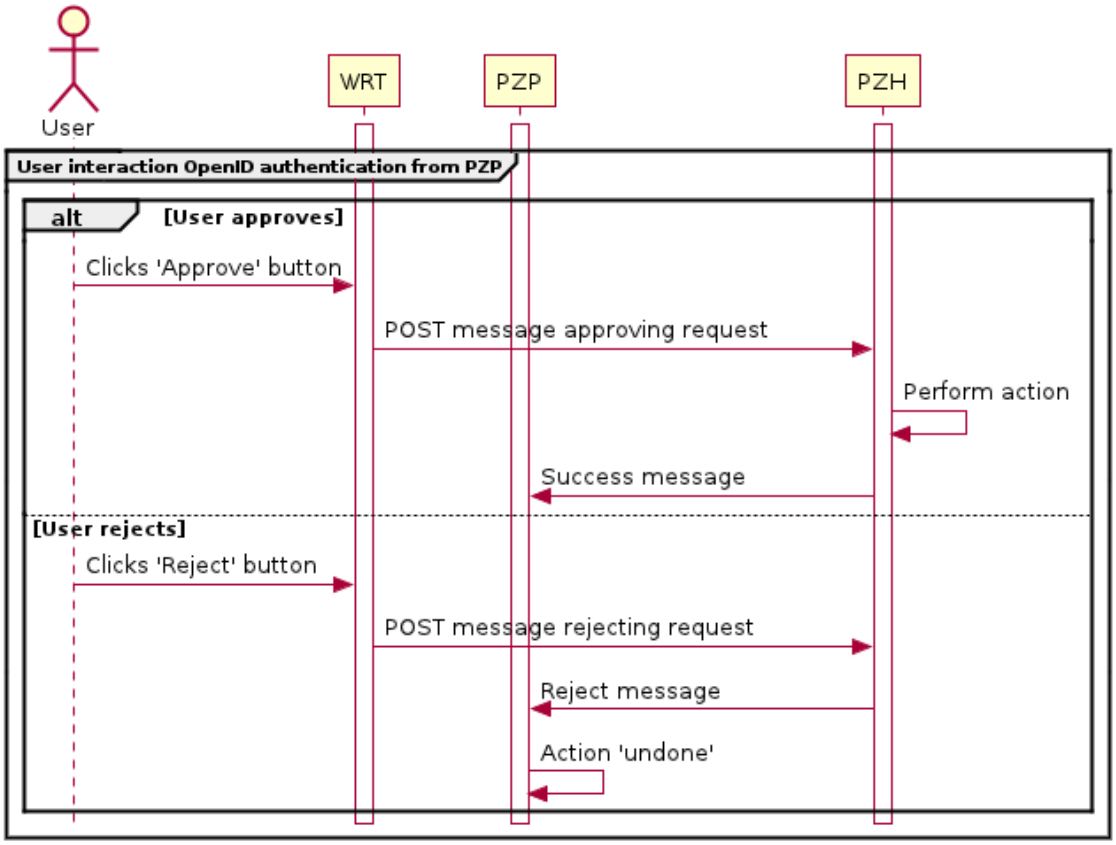


Figure 6: OpenID authentication from the PZP part n.2 - reprinted from [4]

**2.4.1.4 Third party authentication** Third party services should interact with webinos using OAuth 1.0. This method provides a process for end-users to authorize third-party access to their server resources without sharing their credentials, through user-agent redirections.

The solution described consists on the use of an additional developer-provided server (*devServer*) which is trusted to hold on to developer OAuth credentials. This is accessed by the client-side JavaScript of a webinos application in order to sign OAuth requests and gain access to resources. Storing credentials on a server, misuse of these credentials can be prevented through rate-limiting or other mechanisms.

A practical implementation of this is described in Fig. 7.
The diagram shown is related to a webinos application (that is running within the widget runtime (WRT) or browser) attempting to connect to Twitter (or any similar OAuth 1.0 service). The first section describes how the application requests that the *devServer* authenticate to Twitter on the application behalf. Following this, an example of using the *devServer* to make access requests (tweets, in this case) is shown.

### 2.4.2 Security implementation in Authentication methods

As discussed previously, in the webinos authentication method security is implemented in different ways depending on the authentication required.
In the *User authentication* security is guaranteed using the OpenID Provider Authentication Policy Extension (OpenID PAPE) over HTTPS that provides a mechanism by which a Relying Party can request that particular authentication policies be applied by the OpenID Provider when authenticating an End User.

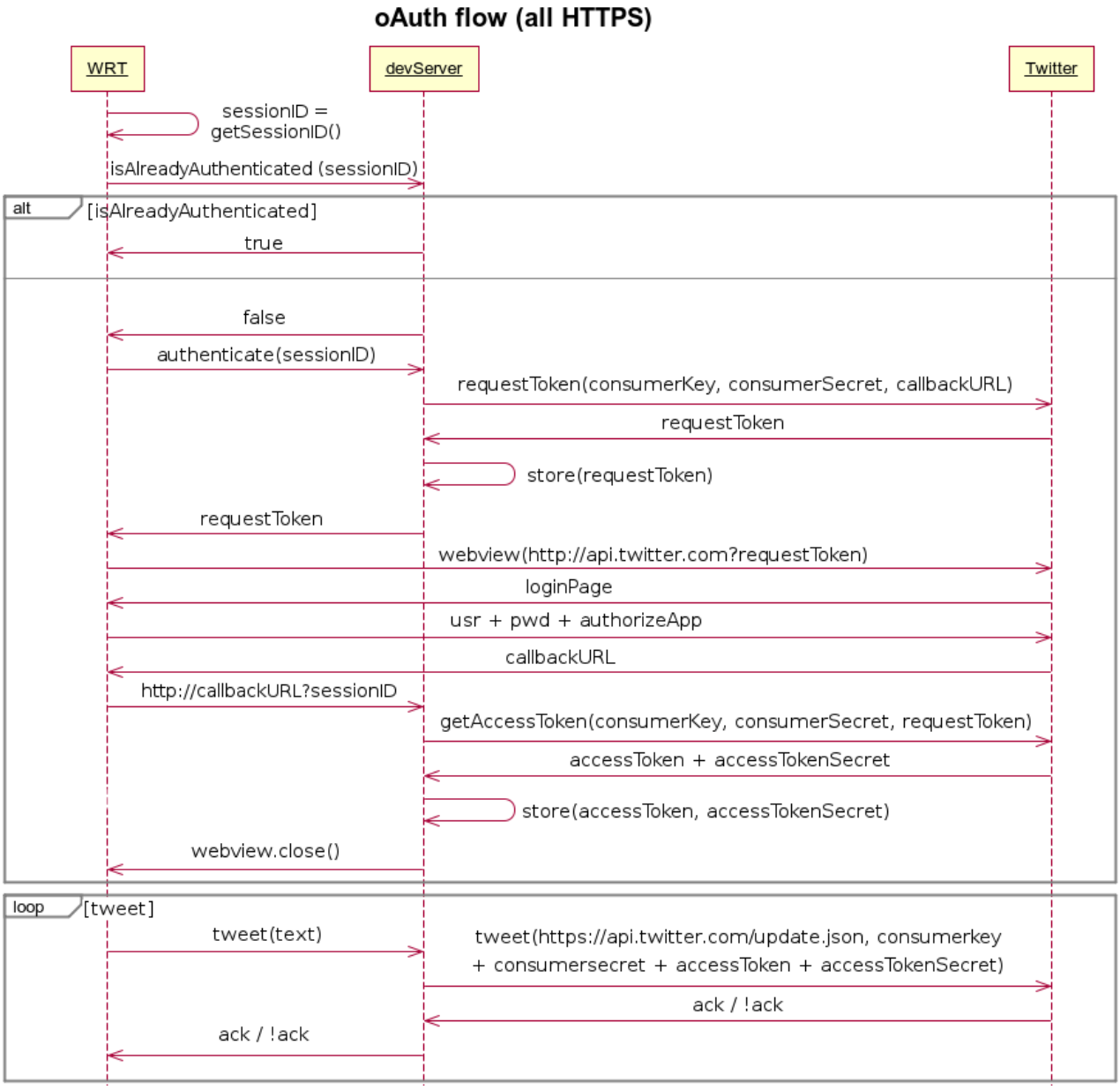


Figure 7: Third party authentication - reprinted from [5]

On the other hand, the security for *Device authentication* and *Third party authentication* is guaranteed using RSA private keys. Through these methods we can guarantee the following security properties:

- Authentication (simple / mutual): depending on the security methods available on the server and on the device, and on the kind of certificates available, every part of the communication can ask each other to demonstrate that they are exactly who they say they are.
- Integrity: if someone intercepts our messages and modifies them, the use of keyed digest and message identifiers (offered by HTTPS) guarantees that the system can detect these modifications.
- Confidentiality: The system cyphers all the data sent by each part of the communication and so, even if someone intercepts them, they cannot understand the content. However this is not an absolute assumption: The use of RSA private key is not an absolute warranty, because the level of security offered by this method depends on the length of the key used to cypher data, on the exponent used for the 2 keys (if the exponent contains a little number of 1, it could be simple to decipher data for an hypothetical attacker) and on the use that the user makes of the private key (if, for example, the user stores the key in a place that is easy accessible for anyone, all efforts are pointless) .
- Non repudiation: Using RSA private keys we can certificate who the owner of every message/data is. However, care must be taken even for this point: if the RSA keys are not generated carefully and the user does not store the private key in an appropriate place, the property becomes false.

Considering these properties we can avoid a lot of security attacks:

- Packet sniffing: Even if someone intercepts our communications, he/she cannot understand what we are saying because in both the situations (user authentication by PAPE methods or using RSA private key) the data are cyphered.
- Data spoofing: Even if someone tries to modify the content of messages the system detects these modifications.
- Replay attacks: Using HTTPS (SSL on HTTP) this attacks are avoided using a sequence number.
- Shadow server: Using RSA private keys, a server authentication for all the communications is always required.
- Man in the middle: Ensuring the properties of authentication, integrity, confidentiality and using sequence numbers this attack is avoided.

The only attack not avoided by the use of these methods is the DOS (Denial of Service) attack and we demonstrate that the system is actually exposed to this type of attacks [1].

Specifically, considering all the methods just described and all the security controls made by webinos, this attack is applicable only in communications between 2 different entities. If we look at the first sequence diagram (Fig. 2) we can identify the situations in which a DoS attack is applicable in the following situations:

[1] This study is based only on the available webinos documentation (see [3] , [4] , [5] for details), so it is possible that in the real implementation of webinos the developers adopt some other kind of solutions to solve this problem.

- When a message is sent from the *webinos javascript libary* to the *PZH webserver* or viceversa (messages n. 3, 6)
- When a message is sent from the *User agent* to the *Identity provider* or viceversa (messages n. 8, 9, 12, 13, 16)
- When a message is sent from the *Identity provider* to the *PZH webserver* or viceversa (messages n. 17)
- When a message is sent from the *PZH webserver* to the *User agent* or viceversa (messages n. 19)

Then, in the diagram at Fig. 3 it is applicable only in the last message sent from the *PZP Session Handler* to the *PZH Provider*.

Moreover, in the diagram at Fig. 4 it is applicable in the following situations:

- When a message is sent from *PZP* to the *OpenID Proxy* or viceversa (messages n. 6, 9, 10, 13)
- When a message is sent from the *User* to the *OpenID Proxy* or viceversa (messages n. 7, 8, 11, 12)

And, at the end, in the diagrams related to the OpenID authentication from the PZP (Fig. 5 and Fig. 6) and in the one related to the third party authentication (Fig. 7) it is applicable in all the situations.

# 3 Model checker

## 3.1 What is a model checker

Software model checking is the algorithmic analysis of programs to prove properties of their execution. Typically, when a software is made, programmers start from the specification and the definition of all the properties that the system should have for the specific task. Before the real implementation of the code is essential to prove that the specifications wrote do not contain deadlocks, critical states, bugs and/or protocol weaknesses that can cause system crashes or some other kind of problem that could put user information at risk (we can think for example to privacy).

In order to solve such a problem algorithmically, both the model of the system and the specification are formulated in some precise mathematical language: To this purpose, it is formulated as a task in logic, namely to check whether a given structure satisfies a given logical formula.

## 3.2 Model checkers analyzed

There are a lot of software developed for model checking, and for this reason it was a little difficult to find the right for our purpose. Considering the time available for this homework we considered the most diffused software:

- Casper [10]: it is a program that takes a description of a security protocol in a simple, abstract language, and produce a CSP description of the same protocol, suitable for checking using FDR2. It can be used either to find attacks upon protocols, or to show that no such attack exists.
- Promela - Spin [11]: Spin is a tool for analyzing the logical consistency of asynchronous systems, specifically distributed software and communication protocols. A verification model of the system is first specified in a guarded command language called Promela. This specification language allows for the modeling of dynamic creation of asynchronous processes, nondeterministic case selection, loops, gotos, local and global variables. It also allows for a concise specification of logical correctness requirements, including, but not restricted to requirements expressed in linear temporal logic.
- NuSMV [13]: it allows for the representation of synchronous and asynchronous finite state systems, and for the analysis of specifications expressed in Computation Tree Logic (CTL) and Linear Temporal Logic (LTL).

## 3.3 Model checker chosen for our project

Let us analyze with more details the positive and negative characteristics of every model checker analyzed.

- Casper:

  Analyzing all the features (positive and negative) shown in the table 1, we can understand that Casper would be the perfect candidate for our purposes, but we had to consider that there were not a lot of examples and tutorials available for it, and moreover, that we had not enough information about the security relative to webinos to use in the definition of the regular grammar.

- Promela - Spin:

  As shown in the table 2, it is easy to see that the reason why we had not chosen this model checker is that there was not enough documentation about the behaviour of Spin, and, moreover, there were not enough projects from which we could take inspiration.

- NuSMV:

  It was the ideal candidate for our purposes because there were a lot of example project usable as references and a detailed documentation explaining exactly how it works. We can see the properties analyzed for our choice in the table 3.

After an hard work of analysis of all the available materials found for each model checker and of what every model checker was able to do, considering the time available and the workload that we should respect for an homework like this, the availability of comprehensive documentation and examples has been a key decision criterion; So we decided to choose the NuSMV model checker, which was the one for which there were a lot of documentation and old example projects to use as reference.

| **Casper** | |
|---|---|
| *PROS* | *CONS* |
| It is specifically created for security tests | We did not have the information about the keys used by the RSA algorithm |
| It uses a simple abstract language | We did not have the information about the authentication methods available on each device (that should be negotiated in the OAuth procedure) |
| | There are not a lot of example projects explaining how to translate a sequence diagram to a formal language acceptable from Casper |

Table 1: PROS and CONS of Casper model checker

| **Promela - Spin** | |
|---|---|
| *PROS* | *CONS* |
| It is specifically created to test distributed software and communication protocols | There are not exhaustive free tutorials and/or manuals |
| It allows to create dynamically asynchronous processes | There are not a lot of example projects explaining how to translate a sequence diagram to a formal language acceptable from Spin |

Table 2: PROS and CONS of Spin model checker

| **NuSMV** | |
|---|---|
| *PROS* | *CONS* |
| It allows for the representation of synchronous and asynchronous finite state systems<br><br>It allows for the analysis of specifications expressed in Computation Tree Logic (CTL) and Linear Temporal Logic (LTL)<br><br>It is possible to use variables (like in a sequence diagram) by which control the execution of the state machine<br><br>There are a lot of examples and projects using NuSMV<br><br>There is an easy understandable manual [12] and an easy understandable tutorial [13] | By the version 2.5.2 NuSMV processes are still supported, but deprecated (to see more details go to the link [15] ) |

Table 3: PROS and CONS of NuSMV model checker

# 4 Properties of the webinos authentication system

We started from the analysis of what the model checker would have to be able to verify.
We had to verify that the authentication methods were not affected by errors and/or deadlocks. For this reasons we decided to concentrate our efforts on the test of the procedures illustrated in the sequence diagrams reported in the first section of this report ( Fig. 2 - Fig. 3 - Fig. 4 - Fig. 5 - Fig. 6 - Fig. 7 ) and to test if, based on the specification, the system was exposed to the DoS attack.
Therefore, the properties to be checked was that all the states of the sequence diagrams were reachable and that even if someone has tried to occupy the system by a DoS attack it does not crash.

## 4.1 Specific model for the webinos authentication system

### 4.1.1 Hypothesis and assumption

ANNOTATION 1: Even if webinos accepts simultaneous asynchronous connection, we decided to simulate only one process at a time. The reasons are two:

- From the version 2.5.2 NuSMV processes are still supported, but deprecated (you can see more details to the link [15] ).
- We assume that webinos has composed by a queue that stores all the messages arriving from each actor and so every process can be treated as unique.

As a result of this assumption we can't analyze all the situations in which a Denial of Service is achieved overloading resources, but it is not a real problem for us because there are not enough information about the amount of concurrent connections supported by each entity and/or the amount of time available for an answer, in the documentation. So, even if we had the possibility to execute more processes simultaneously, we would not have the necessary information to do it.
Moreover, even if we can see more than one process in all the results, they corresponds to the different situations reachable from each state depending on the variable values.

In this first part we will describe how a language accepted by the NuSMV model checker is made [2].

First of all we need to know that all the languages (stored in files with "smv" extension) accepted by the NuSMV model checker has to be composed by MODULEs. The module that has to be present in every language is the "main" module (MODULE main). From it, it is possible to call and "execute" other modules passing variable to them. In each module there are 2 main sections:

- VAR: in which we define the states of the language
- ASSIGN: in which we define the rules followed by the model checker to pass from a state to another and in which we define the properties that must be verified (specified by the word SPEC);

[2] For more details about the languages accepted by NuSMV let us see [12] and [13]

### 4.1.2 How to convert a sequence diagram into a formal language

Now that we know how a language is made up we can pass to see how to define all the parts.

As we have already said, the procedure shown here was repeated for all the 5 diagrams reported in the first section of this report ( Fig. 2 - Fig. 3 - Fig. 4 - (Fig. 5 and Fig. 6 treated as one) - Fig. 7 ).

Let us start from the section VAR. Looking at the first diagram ( Fig. 2 ) we can imagine that all the actors of the diagram are states container and then, every arrow of the diagram (that represents a method invocation or a message sent to the other part) can represents a state assumed by the system.

To better understand what we are saying let us see the diagram of our interest. As shown in the table 4, there are 6 actors and so we created 6 variables.

| ***Actors*** | ***Corresponding variable*** |
|---|---|
| *User* | *User_status* |
| *User agent* | *Useragent_status* |
| *Webinos javascript library* | *Webinos_javascript_library_status* |
| *PZH webserver* | *PZH_webserver__status* |
| *OpenID Module* | *OpenID_Module__status* |
| *Identity provider* | *Identity_provider_status* |

Table 4: Actors and variables corresponding to the first sequence diagram

Then, we created another variable for the system (*system_status*) that can be considered as a general state indicating the point of the diagram in which the exchange of messages is in a particular moment. After that we started to create a state for each arrow present in the diagram:

- *login_request*
- *auth_agent_req*
- *auth_library_req*
- *prepare_openid_auth*
- *openid_url_par_from_openid*
- *openid_url_par_from_pzh*
- *load_auth_url*
- *identity_prov_url*
- *identity_prov_webpage*
- *ask_user_pass*
- *insert_pass*
- *auth2_from_agent*

- *auth_req_from_identity_prov*
- *auth_req_agent*
- *allow_user*
- *allow_agent*
- *pzh_callback*
- *extract_user_attributes*
- *redirect_pzh_landing_page*

These states are all the possible states that the system can assume.
Depending on these states, all the other actors act in different ways and assume different values. Each value represents a particular message sent to other actors, and so we used the same name used in the system status to represent these states. So, we distributed the states shown above to all the actors.
(
For esample, the user will assume the following states:

- *send_login_request*
- *send_right_userpass*
- *send_wrong_userpass*
- *allow_auth*
- *reject_auth*
- *authenticated*

)

Moreover, we have inserted 2 more states with respect to those present in the "system state" variable because we would like to see what would happen when the user types a wrong password or rejects the authorization request.

Obviously we did the same thing for all the actors (so for all the variables) present in the system.

As we can easily understand, by the definition of the states described above it is not possible to simulate a DoS attack and, moreover, it is not possible to express an idle actor, so we added 2 more states to all the actors (even if, as described in *Security implementation in Authentication methods*, not all the interactions could be affected by a DoS attack, so it is important to take care in the next step of the language definition: *Definition of the rules (ASSIGN section)*):

- *idle*
- *busy*

The busy state is the key for the DoS attack: it simulates the situation in which some point of the system are engaged to answer to the attacker.

### Definition of the rules (ASSIGN section)

After the definition of all the states of the system it is important to define the rules that let the system move between states. Let us analyze one of the definition made for the grammar described above:

```
        init(javascript_library_status):= idle;

next(javascript_library_status):= case
                system_status = auth_agent_req & (javascript_library_status
                   = idle | javascript_library_status = busy) :
                    send_auth_req;
                system_status = openid_url_par_from_pzh & (
                   javascript_library_status = idle |
                   javascript_library_status = busy)
                   : {load_auth_url,busy};
                TRUE : idle;
        esac;
```

In this piece of code we say to model checker to initialize the variable *javascript_library_status* to the value *idle* and then we say how its value has to be updated:
when the *system_status* is equal to *auth_agent_req* AND the value of *javascript_library_status* is *idle* or *busy*, the variable *javascript_library_status* has to assume the value *send_auth_req*.
Then, when *system_status* is equal to *openid_url_par_from_pzh* AND the value of *javascript_library_status* is *idle* or *busy*, the variable *javascript_library_status* has to assume the value *load_auth_url* or the value *busy*: as you can see in the diagram, the message *openid_url_par_from_pzh* came from a component of the system that is out of the block that contains the *javascript library*, so in this case there could be a DoS attack and for this reason we say to the model checker that in this situation the *javascript library* can answer immediately (assuming the value *load_auth_url*) or not (assuming the value *busy*).
Then, if neither the first and the second condition are satisfied the value of *javascript_library_status* has to be *idle*.

After that, we made this kind of definition for all the variables and then we had passed to the final part: the definition of the properties that have to be verified.

### Properties to verify (ASSIGN section)

To understand this part let us analyze one of the possible properties:

```
SPEC AG (proc1.system_status = idle -> AF proc1.system_status = login_request)
```

The keyword *SPEC* says that it is a property that has to be verified. The keyword *AG* says that the property has to be valid *globally* and for *all* the states of all the possible path. After that we say that when the system status is *idle* then the next state (*AF* means *All Future*) of *system_status* has to be *login_request*.

Let us see another property:

```
SPEC AG (proc1.user_status = send_login_request -> EF proc1.useragent_status =
    send_auth_req)
```

It says that for *all the states* of *all the possible path* (AG) when the *user_status* is *send_login_request* then one of the possible future state (*EF* means that *Exist* some *Future* path) of *useragent_status* should be *send_auth_req*.

For all the NuSMV code produced for this project let us see to the appendix section in which is described where you can find it.

# 5 Results

## 5.1 Results from the Model checker

The results produced by the model checker tells us if the properties defined are satisfied or not, and if they are not satisfied it shows a counter example that demonstrates it. Let us see this 2 first parts of the results reported in Appendix B:

```
-- specification AG (proc1.system_status = idle -> AF proc1.system_status =
   login_request) is true
-- specification AG (proc1.system_status = login_request -> AF proc1.
   system_status = auth_agent_req) is true
-- specification AG (proc1.system_status = auth_agent_req -> AF proc1.
   system_status = auth_library_req) is true
-- specification AG (proc1.system_status = auth_library_req -> AF proc1.
   system_status = prepare_openid_auth) is false
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
-> State: 1.1 <-
  proc1.system_status = idle
  proc1.user_status = send_login_request
  proc1.useragent_status = idle
  proc1.javascript_library_status = idle
  proc1.pzh_webserver_status = idle
  proc1.openid_module_status = idle
  proc1.identity_provider_status = idle
  proc2.system_status = idle
  proc2.user_status = send_login_request
  proc2.useragent_status = idle
  proc2.javascript_library_status = idle
  proc2.pzh_webserver_status = idle
  proc2.openid_module_status = idle
  proc2.identity_provider_status = idle
  proc3.system_status = idle
  proc3.user_status = send_login_request
  proc3.useragent_status = idle
  proc3.javascript_library_status = idle
  proc3.pzh_webserver_status = idle
  proc3.openid_module_status = idle
  proc3.identity_provider_status = idle
-> State: 1.2 <-
  proc1.system_status = login_request
  proc1.user_status = idle
  proc2.system_status = login_request
  proc2.user_status = idle
  proc3.system_status = login_request
  proc3.user_status = idle
-> State: 1.3 <-
  proc1.useragent_status = send_auth_req
  proc2.useragent_status = send_auth_req
  proc3.useragent_status = send_auth_req
-> State: 1.4 <-
  proc1.system_status = auth_agent_req
```

```
  proc1.useragent_status = idle
  proc2.system_status = auth_agent_req
  proc2.useragent_status = idle
  proc3.system_status = auth_agent_req
  proc3.useragent_status = idle
-> State: 1.5 <-
  proc1.javascript_library_status = send_auth_req
  proc2.javascript_library_status = send_auth_req
  proc3.javascript_library_status = send_auth_req
-> State: 1.6 <-
  proc1.system_status = auth_library_req
  proc1.javascript_library_status = idle
  proc2.system_status = auth_library_req
  proc2.javascript_library_status = idle
  proc3.system_status = auth_library_req
  proc3.javascript_library_status = idle
-- Loop starts here
-> State: 1.7 <-
  proc1.pzh_webserver_status = busy
  proc2.pzh_webserver_status = busy
  proc3.pzh_webserver_status = busy
-> State: 1.8 <-
```

As we can see, the first 3 properties are satisfied, instead the fourth is not.
To be precise, the counterexample says us that that if an intruder tries to occupy the *pzh_webserver* with a DoS attack, the system will crash.

Another interesting example is the following:

```
-- specification AG (proc1.system_status = ask_user_pass -> AF proc1.
   system_status = insert_pass) is false
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
-> State: 4.1 <-
  proc1.system_status = idle
  proc1.user_status = send_login_request
  proc1.useragent_status = idle
  proc1.javascript_library_status = idle
  proc1.pzh_webserver_status = idle
  proc1.openid_module_status = idle
  proc1.identity_provider_status = idle
  proc2.system_status = idle
  proc2.user_status = send_login_request
  proc2.useragent_status = idle
  proc2.javascript_library_status = idle
  proc2.pzh_webserver_status = idle
  proc2.openid_module_status = idle
  proc2.identity_provider_status = idle
  proc3.system_status = idle
  proc3.user_status = send_login_request
  proc3.useragent_status = idle
  proc3.javascript_library_status = idle
  proc3.pzh_webserver_status = idle
  proc3.openid_module_status = idle
  proc3.identity_provider_status = idle
-> State: 4.2 <-
  ...

  ...
```

```
-- Loop starts here
-> State: 4.23 <-
  proc1.user_status = send_wrong_userpass
  proc2.identity_provider_status = busy
  proc3.identity_provider_status = busy
-> State: 4.24 <-
  proc1.user_status = idle
-> State: 4.25 <-
  proc1.user_status = send_wrong_userpass
```

Analyzing the last line of the result shown, it is easy to understand that the system will crash if the user types a wrong password instead of a right one. Moreover, on the other hand, the error obtained in the previous example is already present, in fact just one step before the last passage, the *proc2* and the *proc3* crashes for a DoS attack.

For all the results obtained from the model checker let us see the Appendix B.

## 5.2 Results analysis with suggestions for future development

Analyzing the results obtained from all the formal languages made to represent all the possible transitions in the authentication process, we can see that the system has the following 2 main problems:

- It is exposed to DoS attacks
- The system crashes when the user does something that is not expected and reported in the diagrams

Obviously these considerations are the results made implementing what is reported in the specification. As we can see, there are not information about what happens when, for example, the user inserts a wrong password, or when he rejects an authentication request. Furthermore, we do not know how many time is spent by every actors waiting an answer or a command from other actors, so, as we can see in all the results, assuming that the real system works exactly as described in the documentation, the system could be exposed to a DoS attack.

It is demonstrated in all the counter examples in which the system enter in a loop depending on the fact that from a busy state it could continue to remain in this situation for an infinite time.

Moreover, we can see that every time the user does something that is not expected, the system does not respect the normal sequence of action: it remains in the state caused by the action, so it crashes.

Even if the problems just illustrated are not so trivial, we can observe, on the other hand, that all the other properties are satisfied and so all the other activities are not affected by errors or deadlocks.

# 6 Conclusions

For future works we would like to suggest to the developers to write a more detailed documentation and then, if actually the system does not implement anything to solve the 2 problems exposed in the previous section we can suggest to implement some functions to solve it.

All the code developed for this project and the results obtained by the model checker are attached to this document and is organized as described below.
There is a folder for each diagram analyzed:

- webinos_1_OpenID_Authentication -> related to Fig. 2
- webinos_2_dev_held_priv_key -> related to Fig. 3
- webinos_3_local_dev_auth -> related to Fig. 4
- webinos_4_openid_auth_from_pzp -> related to Fig. 5 and Fig. 6
- webinos_5_oauth_service_twitter -> related to Fig. 7

In each folder we can find a file with "smv" extension, which contains the formal languages and the properties that have to be verified;
and a file with "txt" extension, which contains the results.

# A User manual

In order to use the source code produced for this homework, we have to follow the steps listed below:

- Download NuSMV software from the official site (following this link: [16])
- Choose the right version (looking for the right operating system) and after the download, install it on the computer.
  Let us assume that we are working under a linux distribution, so we can follow the "NuSMV source" link and after entering the required information we can download the "tar.gz" archive.
- Uncompress the content of the archive in a directory of your choice (/temp for example)
- Then, open the terminal window
- Enter the directory in which you have extracted the files (/temp)
- Enter the directory "cudd-2.4.1.1"
  ```
  cd cudd-2.4.1.1
  ```
- Run "*make*"
- Enter the directory "../nusmv"
- Run "configure":
  ```
  ./configure
  ```
- Run "*make*"

Now, let us assume that the formal language defined is in the file "example.smv" and that we want to store the results in the file "results.txt". To let the model checker do it we have to run the following command:

```
NuSMV example.smv > results.txt
```

After a few seconds we will have the results stored in the file "results.txt".

# B NuSMV code implemented for the first sequence diagram (OpenID Authentication)

```
MODULE openid_authentication()
VAR
        system_status:{idle,login_request, auth_agent_req, auth_library_req,
            prepare_openid_auth, openid_url_par_from_openid,
            openid_url_par_from_pzh, load_auth_url, identity_prov_url,
            identity_prov_webpage, ask_user_pass, insert_pass, auth2_from_agent,
            auth_req_from_identity_prov, auth_req_agent, allow_user, allow_agent,
             pzh_callback, extract_user_attributes, redirect_pzh_landing_page};

        user_status:{idle, send_login_request, send_right_userpass,
            send_wrong_userpass, allow_auth, reject_auth,authenticated};

        useragent_status:{idle, send_auth_req, send_identity_provider_url_req,
            ask_userpass, send_second_auth_req, ask_auth, send_granted_auth,
            authenticated};

        javascript_library_status:{idle, busy, send_auth_req, load_auth_url};

        pzh_webserver_status:{idle, busy, send_openid_auth_req,
            send_openid_auth_url_par, extract_openid_attributes,
            send_redirect_to_pzh_land};

        openid_module_status:{idle, busy, send_openid_auth_url_par};

        identity_provider_status:{idle, busy, send_identity_provider_webpage,
            send_auth_req, send_pzh_callback};


ASSIGN

        init(useragent_status):= idle;

        init(javascript_library_status):= idle;

        init(pzh_webserver_status):= idle;

        init(openid_module_status):= idle;

        init(identity_provider_status):= idle;


        next(system_status):= case
                system_status = idle & user_status = send_login_request:
                    login_request;
                system_status = login_request & useragent_status = send_auth_req
                     : auth_agent_req;
                system_status = auth_agent_req  & javascript_library_status =
                    send_auth_req : auth_library_req;
                system_status = auth_library_req & pzh_webserver_status =
                    send_openid_auth_req  : prepare_openid_auth;
                system_status = prepare_openid_auth & openid_module_status =
                    send_openid_auth_url_par : openid_url_par_from_openid;
                system_status = openid_url_par_from_openid &
                    pzh_webserver_status = send_openid_auth_url_par :
                    openid_url_par_from_pzh;
```

```
            system_status = openid_url_par_from_pzh &
               javascript_library_status = load_auth_url : load_auth_url;
            system_status = load_auth_url & useragent_status =
               send_identity_provider_url_req : identity_prov_url;
            system_status = identity_prov_url & identity_provider_status=
               send_identity_provider_webpage : identity_prov_webpage;
            system_status = identity_prov_webpage & useragent_status =
               ask_userpass : ask_user_pass;
            system_status = ask_user_pass & user_status =
               send_right_userpass : insert_pass;
            system_status = insert_pass & useragent_status =
               send_second_auth_req : auth2_from_agent;
            system_status = auth2_from_agent & identity_provider_status =
               send_auth_req : auth_req_from_identity_prov;
            system_status =  auth_req_from_identity_prov & useragent_status
               = ask_auth : auth_req_agent;
            system_status =  auth_req_agent & user_status = allow_auth :
               allow_user;
            system_status =  allow_user & useragent_status =
               send_granted_auth : allow_agent;
            system_status =  allow_agent & identity_provider_status=
               send_pzh_callback : pzh_callback;
            system_status =  pzh_callback & pzh_webserver_status =
               extract_openid_attributes : extract_user_attributes;
            system_status =  extract_user_attributes & pzh_webserver_status
               = send_redirect_to_pzh_land : redirect_pzh_landing_page;
            TRUE : system_status;
    esac;

    next(user_status):= case
            system_status = ask_user_pass & (user_status = idle) : {
               send_right_userpass, send_wrong_userpass};
            system_status = auth_req_agent & (user_status = idle) : {
               allow_auth, reject_auth};
            useragent_status = authenticated & (user_status = idle) :
               authenticated;
            TRUE : idle;

    esac;

    next(useragent_status):= case
            system_status = login_request & useragent_status = idle:
               send_auth_req;
            system_status = load_auth_url & useragent_status = idle:
               send_identity_provider_url_req;
            system_status = identity_prov_webpage & useragent_status = idle:
                ask_userpass;
            system_status = insert_pass & useragent_status = idle:
               send_second_auth_req;
            system_status = auth_req_from_identity_prov & useragent_status =
                idle: ask_auth;
            system_status = allow_user & useragent_status = idle:
               send_granted_auth;
            system_status = redirect_pzh_landing_page & useragent_status =
               idle : authenticated;
            TRUE : idle;
    esac;

    next(javascript_library_status):= case
```

```
                system_status = auth_agent_req & (javascript_library_status =
                    idle | javascript_library_status = busy) : send_auth_req;
                system_status = openid_url_par_from_pzh & (
                    javascript_library_status = idle | javascript_library_status
                    = busy) : {load_auth_url,busy};
                TRUE : idle;
        esac;

        next(pzh_webserver_status):= case
                system_status = auth_library_req & (pzh_webserver_status = idle
                    | pzh_webserver_status = busy) : {send_openid_auth_req,busy};
                system_status = openid_url_par_from_openid & (
                    pzh_webserver_status = idle | pzh_webserver_status = busy) :
                    send_openid_auth_url_par;
                system_status = pzh_callback & (pzh_webserver_status = idle |
                    pzh_webserver_status = busy) : {extract_openid_attributes,
                    busy};
                system_status = extract_user_attributes & (pzh_webserver_status
                    = idle | pzh_webserver_status = busy) :
                    send_redirect_to_pzh_land;
                TRUE : idle;
        esac;

        next(openid_module_status):= case
                system_status = prepare_openid_auth & (openid_module_status =
                    idle | openid_module_status = busy) :
                    send_openid_auth_url_par;
                TRUE : idle;
        esac;

        next(identity_provider_status):= case

                system_status = identity_prov_url & (identity_provider_status =
                    idle | identity_provider_status = busy) : {
                    send_identity_provider_webpage,busy};
                system_status = auth2_from_agent & (identity_provider_status =
                    idle | identity_provider_status = busy) : {send_auth_req,busy
                    };
                system_status = allow_agent & (identity_provider_status = idle |
                     identity_provider_status = busy) : send_pzh_callback;
                TRUE : idle;
        esac;


        -- si può usare solo con asincroni che non verranno più supportati in
            futuro!
        --FAIRNESS
        --      running


MODULE main
VAR

        proc1: openid_authentication();
        proc2: openid_authentication();
        proc3: openid_authentication();

ASSIGN
        init(proc1.user_status) := send_login_request;
        init(proc1.system_status) := idle;
```

```
init(proc2.user_status) := send_login_request;
        init(proc2.system_status) := idle;


init(proc3.user_status) := send_login_request;
        init(proc3.system_status) := idle;


        -- questo dovrebbe essere un commento
        SPEC AG (proc1.system_status = idle -> AF proc1.system_status =
            login_request)   --è sempre verificato globalmente che quando lo
            stato system_status è idle allora il prossimo stato futuro di
            system_status è login_request!
        SPEC AG (proc1.system_status = login_request -> AF proc1.system_status =
             auth_agent_req)
        SPEC AG (proc1.system_status = auth_agent_req -> AF proc1.system_status
            = auth_library_req)
        SPEC AG (proc1.system_status = auth_library_req -> AF proc1.
            system_status = prepare_openid_auth)
        SPEC AG (proc1.system_status = prepare_openid_auth -> AF proc1.
            system_status = openid_url_par_from_openid)


        SPEC AG (proc1.system_status = openid_url_par_from_openid -> AF proc1.
            system_status = openid_url_par_from_pzh)
        SPEC AG (proc1.system_status = openid_url_par_from_pzh -> AF proc1.
            system_status = load_auth_url)
        SPEC AG (proc1.system_status = load_auth_url -> AF proc1.system_status =
             identity_prov_url)
        SPEC AG (proc1.system_status = identity_prov_url -> AF proc1.
            system_status = identity_prov_webpage)
        SPEC AG (proc1.system_status = identity_prov_webpage -> AF proc1.
            system_status = ask_user_pass)
        SPEC AG (proc1.system_status = ask_user_pass -> AF proc1.system_status =
             insert_pass)
        SPEC AG (proc1.system_status = insert_pass -> AF proc1.system_status =
            auth2_from_agent)
        SPEC AG (proc1.system_status = auth2_from_agent -> AF proc1.
            system_status = auth_req_from_identity_prov)

        SPEC AG (proc1.system_status = auth_req_from_identity_prov -> AF proc1.
            system_status = auth_req_agent)
        SPEC AG (proc1.system_status = auth_req_agent -> AF proc1.system_status
            = allow_user)
        SPEC AG (proc1.system_status = allow_user -> AF proc1.system_status =
            allow_agent)
        SPEC AG (proc1.system_status = allow_agent -> AF proc1.system_status =
            pzh_callback)
        SPEC AG (proc1.system_status = pzh_callback -> AF proc1.system_status =
            extract_user_attributes)
        SPEC AG (proc1.system_status = extract_user_attributes -> AF proc1.
            system_status = redirect_pzh_landing_page)

SPEC AG (proc1.user_status = send_login_request -> EF proc1.useragent_status =
    send_auth_req)   --è verificato globalmente almeno una volta che quando lo
    stato userstatus è send_login_request allora uno dei prossimi stati futuri di
     useragent_status è send_auth_req!
```

```
SPEC AG (proc1.useragent_status = send_auth_req -> EF proc1.
    javascript_library_status = send_auth_req)
SPEC AG (proc1.javascript_library_status = send_auth_req -> EF proc1.
    pzh_webserver_status = send_openid_auth_req)
SPEC AG (proc1.pzh_webserver_status = send_openid_auth_req -> EF proc1.
    openid_module_status = send_openid_auth_url_par)
SPEC AG (proc1.openid_module_status = send_openid_auth_url_par -> EF proc1.
    pzh_webserver_status = send_openid_auth_url_par)
SPEC AG (proc1.pzh_webserver_status = send_openid_auth_url_par -> EF proc1.
    javascript_library_status = load_auth_url)
SPEC AG (proc1.javascript_library_status = load_auth_url -> EF proc1.
    useragent_status = send_identity_provider_url_req)
SPEC AG (proc1.useragent_status = send_identity_provider_url_req -> EF proc1.
    identity_provider_status = send_identity_provider_webpage)
SPEC AG (proc1.identity_provider_status = send_identity_provider_webpage -> EF
    proc1.useragent_status = ask_userpass)
SPEC AG (proc1.useragent_status = ask_userpass -> EF (proc1.user_status =
    send_right_userpass | proc1.user_status = send_wrong_userpass))
SPEC AG (proc1.user_status = send_right_userpass -> EF proc1.useragent_status =
    send_second_auth_req)
SPEC AG (proc1.useragent_status = send_second_auth_req -> EF proc1.
    identity_provider_status = send_auth_req)
SPEC AG (proc1.identity_provider_status = send_auth_req -> EF proc1.
    useragent_status = ask_auth)
SPEC AG (proc1.useragent_status = ask_auth -> EF (proc1.user_status = allow_auth
     | proc1.user_status = reject_auth))
SPEC AG (proc1.user_status = allow_auth -> EF proc1.useragent_status =
    send_granted_auth)
SPEC AG (proc1.useragent_status = send_granted_auth -> EF proc1.
    identity_provider_status = send_pzh_callback)
SPEC AG (proc1.identity_provider_status = send_pzh_callback -> EF proc1.
    pzh_webserver_status = extract_openid_attributes)
SPEC AG (proc1.pzh_webserver_status = extract_openid_attributes -> EF proc1.
    pzh_webserver_status = send_redirect_to_pzh_land)
SPEC AG (proc1.pzh_webserver_status = send_redirect_to_pzh_land -> EF proc1.
    useragent_status = authenticated)
SPEC AG (proc1.useragent_status = authenticated -> EF proc1.user_status =
    authenticated)

SPEC AG (proc1.user_status = send_wrong_userpass -> EF proc1.useragent_status =
    ask_userpass) -- se inserisce la password sbagliata dovrebbe chiedergli di
    reinserirla
SPEC AG (proc1.user_status = reject_auth -> EF proc1.useragent_status = ask_auth
    )
```

```
        -- questo dovrebbe essere un commento
        SPEC AG (proc2.system_status = idle -> AF proc2.system_status =
            login_request)   --è sempre verificato globalmente che quando lo
            stato system_status è idle allora il prossimo stato futuro di
            system_status è login_request!
        SPEC AG (proc2.system_status = login_request -> AF proc2.system_status =
             auth_agent_req)
        SPEC AG (proc2.system_status = auth_agent_req -> AF proc2.system_status
            = auth_library_req)
```

```
        SPEC AG (proc2.system_status = auth_library_req -> AF proc2.
            system_status = prepare_openid_auth)
        SPEC AG (proc2.system_status = prepare_openid_auth -> AF proc2.
            system_status = openid_url_par_from_openid)


        SPEC AG (proc2.system_status = openid_url_par_from_openid -> AF proc2.
            system_status = openid_url_par_from_pzh)
        SPEC AG (proc2.system_status = openid_url_par_from_pzh -> AF proc2.
            system_status = load_auth_url)
        SPEC AG (proc2.system_status = load_auth_url -> AF proc2.system_status =
             identity_prov_url)
        SPEC AG (proc2.system_status = identity_prov_url -> AF proc2.
            system_status = identity_prov_webpage)
        SPEC AG (proc2.system_status = identity_prov_webpage -> AF proc2.
            system_status = ask_user_pass)
        SPEC AG (proc2.system_status = ask_user_pass -> AF proc2.system_status =
             insert_pass)
        SPEC AG (proc2.system_status = insert_pass -> AF proc2.system_status =
            auth2_from_agent)
        SPEC AG (proc2.system_status = auth2_from_agent -> AF proc2.
            system_status = auth_req_from_identity_prov)
        SPEC AG (proc2.system_status = auth_req_from_identity_prov -> AF proc2.
            system_status = auth_req_agent)
        SPEC AG (proc2.system_status = auth_req_agent -> AF proc2.system_status
            = allow_user)
        SPEC AG (proc2.system_status = allow_user -> AF proc2.system_status =
            allow_agent)
        SPEC AG (proc2.system_status = allow_agent -> AF proc2.system_status =
            pzh_callback)
        SPEC AG (proc2.system_status = pzh_callback -> AF proc2.system_status =
            extract_user_attributes)
        SPEC AG (proc2.system_status = extract_user_attributes -> AF proc2.
            system_status = redirect_pzh_landing_page)

SPEC AG (proc2.user_status = send_login_request -> EF proc2.useragent_status =
    send_auth_req)   --è verificato globalmente almeno una volta che quando lo
    stato userstatus è send_login_request allora uno dei prossimi stati futuri di
     useragent_status è send_auth_req!
SPEC AG (proc2.useragent_status = send_auth_req -> EF proc2.
    javascript_library_status = send_auth_req)
SPEC AG (proc2.javascript_library_status = send_auth_req -> EF proc2.
    pzh_webserver_status = send_openid_auth_req)
SPEC AG (proc2.pzh_webserver_status = send_openid_auth_req -> EF proc2.
    openid_module_status = send_openid_auth_url_par)
SPEC AG (proc2.openid_module_status = send_openid_auth_url_par -> EF proc2.
    pzh_webserver_status = send_openid_auth_url_par)
SPEC AG (proc2.pzh_webserver_status = send_openid_auth_url_par -> EF proc2.
    javascript_library_status = load_auth_url)
SPEC AG (proc2.javascript_library_status = load_auth_url -> EF proc2.
    useragent_status = send_identity_provider_url_req)
SPEC AG (proc2.useragent_status = send_identity_provider_url_req -> EF proc2.
    identity_provider_status = send_identity_provider_webpage)
SPEC AG (proc2.identity_provider_status = send_identity_provider_webpage -> EF
    proc2.useragent_status = ask_userpass)
SPEC AG (proc2.useragent_status = ask_userpass -> EF (proc2.user_status =
    send_right_userpass | proc2.user_status = send_wrong_userpass))
SPEC AG (proc2.user_status = send_right_userpass -> EF proc2.useragent_status =
    send_second_auth_req)
```

```
SPEC AG (proc2.useragent_status = send_second_auth_req -> EF proc2.
    identity_provider_status = send_auth_req)
SPEC AG (proc2.identity_provider_status = send_auth_req -> EF proc2.
    useragent_status = ask_auth)
SPEC AG (proc2.useragent_status = ask_auth -> EF (proc2.user_status = allow_auth
     | proc2.user_status = reject_auth))
SPEC AG (proc2.user_status = allow_auth -> EF proc2.useragent_status =
    send_granted_auth)
SPEC AG (proc2.useragent_status = send_granted_auth -> EF proc2.
    identity_provider_status = send_pzh_callback)
SPEC AG (proc2.identity_provider_status = send_pzh_callback -> EF proc2.
    pzh_webserver_status = extract_openid_attributes)
SPEC AG (proc2.pzh_webserver_status = extract_openid_attributes -> EF proc2.
    pzh_webserver_status = send_redirect_to_pzh_land)
SPEC AG (proc2.pzh_webserver_status = send_redirect_to_pzh_land -> EF proc2.
    useragent_status = authenticated)
SPEC AG (proc2.useragent_status = authenticated -> EF proc2.user_status =
    authenticated)

SPEC AG (proc2.user_status = send_wrong_userpass -> EF proc2.useragent_status =
    ask_userpass) -- se inserisce la password sbagliata dovrebbe chiedergli di
    reinserirla
SPEC AG (proc2.user_status = reject_auth -> EF proc2.useragent_status = ask_auth
    )


        -- questo dovrebbe essere un commento
        SPEC AG (proc3.system_status = idle -> AF proc3.system_status =
            login_request)   --è sempre verificato globalmente che quando lo
            stato system_status è idle allora il prossimo stato futuro di
            system_status è login_request!
        SPEC AG (proc3.system_status = login_request -> AF proc3.system_status =
             auth_agent_req)
        SPEC AG (proc3.system_status = auth_agent_req -> AF proc3.system_status
            = auth_library_req)
        SPEC AG (proc3.system_status = auth_library_req -> AF proc3.
            system_status = prepare_openid_auth)
        SPEC AG (proc3.system_status = prepare_openid_auth -> AF proc3.
            system_status = openid_url_par_from_openid)


        SPEC AG (proc3.system_status = openid_url_par_from_openid -> AF proc3.
            system_status = openid_url_par_from_pzh)
        SPEC AG (proc3.system_status = openid_url_par_from_pzh -> AF proc3.
            system_status = load_auth_url)
        SPEC AG (proc3.system_status = load_auth_url -> AF proc3.system_status =
             identity_prov_url)
        SPEC AG (proc3.system_status = identity_prov_url -> AF proc3.
            system_status = identity_prov_webpage)
        SPEC AG (proc3.system_status = identity_prov_webpage -> AF proc3.
            system_status = ask_user_pass)
        SPEC AG (proc3.system_status = ask_user_pass -> AF proc3.system_status =
             insert_pass)
        SPEC AG (proc3.system_status = insert_pass -> AF proc3.system_status =
            auth2_from_agent)
        SPEC AG (proc3.system_status = auth2_from_agent -> AF proc3.
            system_status = auth_req_from_identity_prov)
        SPEC AG (proc3.system_status = auth_req_from_identity_prov -> AF proc3.
            system_status = auth_req_agent)
```

```
        SPEC AG (proc3.system_status = auth_req_agent -> AF proc3.system_status
            = allow_user)
        SPEC AG (proc3.system_status = allow_user -> AF proc3.system_status =
            allow_agent)
        SPEC AG (proc3.system_status = allow_agent -> AF proc3.system_status =
            pzh_callback)
        SPEC AG (proc3.system_status = pzh_callback -> AF proc3.system_status =
            extract_user_attributes)
        SPEC AG (proc3.system_status = extract_user_attributes -> AF proc3.
            system_status = redirect_pzh_landing_page)

SPEC AG (proc3.user_status = send_login_request -> EF proc3.useragent_status =
    send_auth_req)   --è verificato globalmente almeno una volta che quando lo
    stato userstatus è send_login_request allora uno dei prossimi stati futuri di
     useragent_status è send_auth_req!
SPEC AG (proc3.useragent_status = send_auth_req -> EF proc3.
    javascript_library_status = send_auth_req)
SPEC AG (proc3.javascript_library_status = send_auth_req -> EF proc3.
    pzh_webserver_status = send_openid_auth_req)
SPEC AG (proc3.pzh_webserver_status = send_openid_auth_req -> EF proc3.
    openid_module_status = send_openid_auth_url_par)
SPEC AG (proc3.openid_module_status = send_openid_auth_url_par -> EF proc3.
    pzh_webserver_status = send_openid_auth_url_par)
SPEC AG (proc3.pzh_webserver_status = send_openid_auth_url_par -> EF proc3.
    javascript_library_status = load_auth_url)
SPEC AG (proc3.javascript_library_status = load_auth_url -> EF proc3.
    useragent_status = send_identity_provider_url_req)
SPEC AG (proc3.useragent_status = send_identity_provider_url_req -> EF proc3.
    identity_provider_status = send_identity_provider_webpage)
SPEC AG (proc3.identity_provider_status = send_identity_provider_webpage -> EF
    proc3.useragent_status = ask_userpass)
SPEC AG (proc3.useragent_status = ask_userpass -> EF (proc3.user_status =
    send_right_userpass | proc3.user_status = send_wrong_userpass))
SPEC AG (proc3.user_status = send_right_userpass -> EF proc3.useragent_status =
    send_second_auth_req)
SPEC AG (proc3.useragent_status = send_second_auth_req -> EF proc3.
    identity_provider_status = send_auth_req)
SPEC AG (proc3.identity_provider_status = send_auth_req -> EF proc3.
    useragent_status = ask_auth)
SPEC AG (proc3.useragent_status = ask_auth -> EF (proc3.user_status = allow_auth
     | proc3.user_status = reject_auth))
SPEC AG (proc3.user_status = allow_auth -> EF proc3.useragent_status =
    send_granted_auth)
SPEC AG (proc3.useragent_status = send_granted_auth -> EF proc3.
    identity_provider_status = send_pzh_callback)
SPEC AG (proc3.identity_provider_status = send_pzh_callback -> EF proc3.
    pzh_webserver_status = extract_openid_attributes)
SPEC AG (proc3.pzh_webserver_status = extract_openid_attributes -> EF proc3.
    pzh_webserver_status = send_redirect_to_pzh_land)
SPEC AG (proc3.pzh_webserver_status = send_redirect_to_pzh_land -> EF proc3.
    useragent_status = authenticated)
SPEC AG (proc3.useragent_status = authenticated -> EF proc3.user_status =
    authenticated)

SPEC AG (proc3.user_status = send_wrong_userpass -> EF proc3.useragent_status =
    ask_userpass) -- se inserisce la password sbagliata dovrebbe chiedergli di
    reinserirla
SPEC AG (proc3.user_status = reject_auth -> EF proc3.useragent_status = ask_auth
    )
```

# C First part of Results obtained by the first model checker (OpenID Authentication)

```
*** This is NuSMV 2.5.4 (compiled on Sun Jun 23 21:07:14 UTC 2013)
*** Enabled addons are: compass
*** For more information on NuSMV see <http://nusmv.fbk.eu>
*** or email to <nusmv-users@list.fbk.eu>.
*** Please report bugs to <nusmv-users@fbk.eu>

*** Copyright (c) 2010, Fondazione Bruno Kessler

*** This version of NuSMV is linked to the CUDD library version 2.4.1
*** Copyright (c) 1995-2004, Regents of the University of Colorado

-- specification AG (proc1.system_status = idle -> AF proc1.system_status =
   login_request)  is true
-- specification AG (proc1.system_status = login_request -> AF proc1.
   system_status = auth_agent_req)  is true
-- specification AG (proc1.system_status = auth_agent_req -> AF proc1.
   system_status = auth_library_req)  is true
-- specification AG (proc1.system_status = auth_library_req -> AF proc1.
   system_status = prepare_openid_auth)  is false
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
-> State: 1.1 <-
  proc1.system_status = idle
  proc1.user_status = send_login_request
  proc1.useragent_status = idle
  proc1.javascript_library_status = idle
  proc1.pzh_webserver_status = idle
  proc1.openid_module_status = idle
  proc1.identity_provider_status = idle
  proc2.system_status = idle
  proc2.user_status = send_login_request
  proc2.useragent_status = idle
  proc2.javascript_library_status = idle
  proc2.pzh_webserver_status = idle
  proc2.openid_module_status = idle
  proc2.identity_provider_status = idle
  proc3.system_status = idle
  proc3.user_status = send_login_request
  proc3.useragent_status = idle
  proc3.javascript_library_status = idle
  proc3.pzh_webserver_status = idle
  proc3.openid_module_status = idle
  proc3.identity_provider_status = idle
-> State: 1.2 <-
  proc1.system_status = login_request
  proc1.user_status = idle
  proc2.system_status = login_request
  proc2.user_status = idle
  proc3.system_status = login_request
  proc3.user_status = idle
-> State: 1.3 <-
  proc1.useragent_status = send_auth_req
  proc2.useragent_status = send_auth_req
  proc3.useragent_status = send_auth_req
-> State: 1.4 <-
```

```
  proc1.system_status = auth_agent_req
  proc1.useragent_status = idle
  proc2.system_status = auth_agent_req
  proc2.useragent_status = idle
  proc3.system_status = auth_agent_req
  proc3.useragent_status = idle
-> State: 1.5 <-
  proc1.javascript_library_status = send_auth_req
  proc2.javascript_library_status = send_auth_req
  proc3.javascript_library_status = send_auth_req
-> State: 1.6 <-
  proc1.system_status = auth_library_req
  proc1.javascript_library_status = idle
  proc2.system_status = auth_library_req
  proc2.javascript_library_status = idle
  proc3.system_status = auth_library_req
  proc3.javascript_library_status = idle
-- Loop starts here
-> State: 1.7 <-
  proc1.pzh_webserver_status = busy
  proc2.pzh_webserver_status = busy
  proc3.pzh_webserver_status = busy
-> State: 1.8 <-
-- specification AG (proc1.system_status = prepare_openid_auth -> AF proc1.
   system_status = openid_url_par_from_openid)  is true
-- specification AG (proc1.system_status = openid_url_par_from_openid -> AF
   proc1.system_status = openid_url_par_from_pzh)  is true
-- specification AG (proc1.system_status = openid_url_par_from_pzh -> AF proc1.
   system_status = load_auth_url)  is false
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
-> State: 2.1 <-
  proc1.system_status = idle
  proc1.user_status = send_login_request
  proc1.useragent_status = idle
  proc1.javascript_library_status = idle
  proc1.pzh_webserver_status = idle
  proc1.openid_module_status = idle
  proc1.identity_provider_status = idle
  proc2.system_status = idle
  proc2.user_status = send_login_request
  proc2.useragent_status = idle
  proc2.javascript_library_status = idle
  proc2.pzh_webserver_status = idle
  proc2.openid_module_status = idle
  proc2.identity_provider_status = idle
  proc3.system_status = idle
  proc3.user_status = send_login_request
  proc3.useragent_status = idle
  proc3.javascript_library_status = idle
  proc3.pzh_webserver_status = idle
  proc3.openid_module_status = idle
  proc3.identity_provider_status = idle
-> State: 2.2 <-
  proc1.system_status = login_request
  proc1.user_status = idle
  proc2.system_status = login_request
  proc2.user_status = idle
  proc3.system_status = login_request
```

```
  proc3.user_status = idle
-> State: 2.3 <-
  proc1.useragent_status = send_auth_req
  proc2.useragent_status = send_auth_req
  proc3.useragent_status = send_auth_req
-> State: 2.4 <-
  proc1.system_status = auth_agent_req
  proc1.useragent_status = idle
  proc2.system_status = auth_agent_req
  proc2.useragent_status = idle
  proc3.system_status = auth_agent_req
  proc3.useragent_status = idle
-> State: 2.5 <-
  proc1.javascript_library_status = send_auth_req
  proc2.javascript_library_status = send_auth_req
  proc3.javascript_library_status = send_auth_req
-> State: 2.6 <-
  proc1.system_status = auth_library_req
  proc1.javascript_library_status = idle
  proc2.system_status = auth_library_req
  proc2.javascript_library_status = idle
  proc3.system_status = auth_library_req
  proc3.javascript_library_status = idle
-> State: 2.7 <-
  proc1.pzh_webserver_status = send_openid_auth_req
  proc2.pzh_webserver_status = busy
  proc3.pzh_webserver_status = busy
-> State: 2.8 <-
  proc1.system_status = prepare_openid_auth
  proc1.pzh_webserver_status = idle
-> State: 2.9 <-
  proc1.openid_module_status = send_openid_auth_url_par
-> State: 2.10 <-
  proc1.system_status = openid_url_par_from_openid
  proc1.openid_module_status = idle
-> State: 2.11 <-
  proc1.pzh_webserver_status = send_openid_auth_url_par
-> State: 2.12 <-
  proc1.system_status = openid_url_par_from_pzh
  proc1.pzh_webserver_status = idle
-- Loop starts here
-> State: 2.13 <-
  proc1.javascript_library_status = busy
-> State: 2.14 <-
-- specification AG (proc1.system_status = load_auth_url -> AF proc1.
   system_status = identity_prov_url)  is true
-- specification AG (proc1.system_status = identity_prov_url -> AF proc1.
   system_status = identity_prov_webpage)  is false
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
-> State: 3.1 <-
  proc1.system_status = idle
  proc1.user_status = send_login_request
  proc1.useragent_status = idle
  proc1.javascript_library_status = idle
  proc1.pzh_webserver_status = idle
  proc1.openid_module_status = idle
  proc1.identity_provider_status = idle
  proc2.system_status = idle
```

```
  proc2.user_status = send_login_request
  proc2.useragent_status = idle
  proc2.javascript_library_status = idle
  proc2.pzh_webserver_status = idle
  proc2.openid_module_status = idle
  proc2.identity_provider_status = idle
  proc3.system_status = idle
  proc3.user_status = send_login_request
  proc3.useragent_status = idle
  proc3.javascript_library_status = idle
  proc3.pzh_webserver_status = idle
  proc3.openid_module_status = idle
  proc3.identity_provider_status = idle
-> State: 3.2 <-
  proc1.system_status = login_request
  proc1.user_status = idle
  proc2.system_status = login_request
  proc2.user_status = idle
  proc3.system_status = login_request
  proc3.user_status = idle
-> State: 3.3 <-
  proc1.useragent_status = send_auth_req
  proc2.useragent_status = send_auth_req
  proc3.useragent_status = send_auth_req
-> State: 3.4 <-
  proc1.system_status = auth_agent_req
  proc1.useragent_status = idle
  proc2.system_status = auth_agent_req
  proc2.useragent_status = idle
  proc3.system_status = auth_agent_req
  proc3.useragent_status = idle
-> State: 3.5 <-
  proc1.javascript_library_status = send_auth_req
  proc2.javascript_library_status = send_auth_req
  proc3.javascript_library_status = send_auth_req
-> State: 3.6 <-
  proc1.system_status = auth_library_req
  proc1.javascript_library_status = idle
  proc2.system_status = auth_library_req
  proc2.javascript_library_status = idle
  proc3.system_status = auth_library_req
  proc3.javascript_library_status = idle
-> State: 3.7 <-
  proc1.pzh_webserver_status = send_openid_auth_req
  proc2.pzh_webserver_status = busy
  proc3.pzh_webserver_status = busy
-> State: 3.8 <-
  proc1.system_status = prepare_openid_auth
  proc1.pzh_webserver_status = idle
-> State: 3.9 <-
  proc1.openid_module_status = send_openid_auth_url_par
  proc2.pzh_webserver_status = send_openid_auth_req
  proc3.pzh_webserver_status = send_openid_auth_req
-> State: 3.10 <-
  proc1.system_status = openid_url_par_from_openid
  proc1.openid_module_status = idle
  proc2.system_status = prepare_openid_auth
  proc2.pzh_webserver_status = idle
  proc3.system_status = prepare_openid_auth
  proc3.pzh_webserver_status = idle
```

```
-> State: 3.11 <-
  proc1.pzh_webserver_status = send_openid_auth_url_par
  proc2.openid_module_status = send_openid_auth_url_par
  proc3.openid_module_status = send_openid_auth_url_par
-> State: 3.12 <-
  proc1.system_status = openid_url_par_from_pzh
  proc1.pzh_webserver_status = idle
  proc2.system_status = openid_url_par_from_openid
  proc2.openid_module_status = idle
  proc3.system_status = openid_url_par_from_openid
  proc3.openid_module_status = idle
-> State: 3.13 <-
  proc1.javascript_library_status = load_auth_url
  proc2.pzh_webserver_status = send_openid_auth_url_par
  proc3.pzh_webserver_status = send_openid_auth_url_par
-> State: 3.14 <-
  proc1.system_status = load_auth_url
  proc1.javascript_library_status = idle
  proc2.system_status = openid_url_par_from_pzh
  proc2.pzh_webserver_status = idle
  proc3.system_status = openid_url_par_from_pzh
  proc3.pzh_webserver_status = idle
-> State: 3.15 <-
  proc1.useragent_status = send_identity_provider_url_req
  proc2.javascript_library_status = load_auth_url
  proc3.javascript_library_status = load_auth_url
-> State: 3.16 <-
  proc1.system_status = identity_prov_url
  proc1.useragent_status = idle
  proc2.system_status = load_auth_url
  proc2.javascript_library_status = idle
  proc3.system_status = load_auth_url
  proc3.javascript_library_status = idle
-> State: 3.17 <-
  proc1.identity_provider_status = busy
  proc2.useragent_status = send_identity_provider_url_req
  proc3.useragent_status = send_identity_provider_url_req
-> State: 3.18 <-
  proc2.system_status = identity_prov_url
  proc2.useragent_status = idle
  proc3.system_status = identity_prov_url
  proc3.useragent_status = idle
-- Loop starts here
-> State: 3.19 <-
  proc2.identity_provider_status = busy
  proc3.identity_provider_status = busy
-> State: 3.20 <-
-- specification AG (proc1.system_status = identity_prov_webpage -> AF proc1.
   system_status = ask_user_pass)  is true
-- specification AG (proc1.system_status = ask_user_pass -> AF proc1.
   system_status = insert_pass)  is false
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
-> State: 4.1 <-
  proc1.system_status = idle
  proc1.user_status = send_login_request
  proc1.useragent_status = idle
  proc1.javascript_library_status = idle
  proc1.pzh_webserver_status = idle
```

```
proc1.openid_module_status = idle
proc1.identity_provider_status = idle
proc2.system_status = idle
proc2.user_status = send_login_request
proc2.useragent_status = idle
proc2.javascript_library_status = idle
proc2.pzh_webserver_status = idle
proc2.openid_module_status = idle
proc2.identity_provider_status = idle
proc3.system_status = idle
proc3.user_status = send_login_request
proc3.useragent_status = idle
proc3.javascript_library_status = idle
proc3.pzh_webserver_status = idle
proc3.openid_module_status = idle
```

# References


[1] https://developer.webinos.org/introduction, "Introduction to webinos",

[2] http://www.webinos.org/blog/2011/11/04/what-is-webinos-%E2%80%93-and-why-do-i-want-one/ , "What is webinos - and why do I want one?",

[3] http://www.webinos.org/content/html/D033/Architecture_Overview.htm , "Webinos architecture",

[4] http://www.webinos.org/content/html/D033/Authentication.htm , "Authentication in webinos",

[5] http://www.webinos.org/content/html/D033/OAuth_Service.htm , "OAuth Service -> Authentication by third party application",

[6] http://openid.net/specs/openid-provider-authentication-policy-extension-1_0.html , "OpenID Provider Authentication Policy Extension",

[7] https://en.wikipedia.org/wiki/Model_checking , "An introduction to Model checking",

[8] A. Fantechi, S. Gnesi, "Model Checking: cos'e' e come si applica", Journal "Mondo Digitale", No. 2-3, June - September 2011, pp. 29–39

[9] P.A. Abdulla, B. Jonsson, M. Nilsson,M. Saksena, "A Survey of Regular Model Checking", pp. 1–14

[10] http://www.cs.ox.ac.uk/gavin.lowe/Security/Casper/index.html , "Casper model checker",

[11] http://spinroot.com/spin/Man/ , "Spin model checker",

[12] http://nusmv.fbk.eu/NuSMV/userman/v25/nusmv.pdf , "NuSMV manual",

[13] http://nusmv.fbk.eu/NuSMV/tutorial/v25/tutorial.pdf , "NuSMV tutorial",

[14] M. Panti, L. Spalazzi, S. Tacconi "Document from which we learn to translate a sequence diagram to a formal language: Using the NuSMV Model Checker to verify the Kerberos Protocol", pp. 1–7

[15] http://nusmv.fbk.eu/faq.html#012 , “Why processes are deprecated in NuSMV”,

[16] http://nusmv.fbk.eu/NuSMV/download/getting-v2.html , “NuSMV program”,