Shiva Bhattarai

DEVELOPMENT OF A SECURITY FRAMEWORK FOR HTML5-BASED MOBILE AGENTS

Master of Science Thesis

Examiner: Prof. Kari Juhani Systä
Examiner and topic approved by the Faculty Council of the Faculty of Computing and Electrical Engineering on 4^{th} *November 2015.*

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Information Technology
SHIVA BHATTARAI: Development of a security framework for HTML5-based mobile agents
Master of Science Thesis, 44 pages
April 2015
Major: Software Engineering
Examiner: Prof. Kari Juhani Systä
Keywords: HTML5, mobile agent, agent platform, framework, security, signature, origin server, source address, RSA.

Mobile agent technology is a paradigm where a program can move autonomously in the different executable environment of a network. The program is the mobile agent, that can move its code, suspend and resume the execution in the new environment.The use of a mobile agent provides numerous benefits over the traditional paradigm like client-server. It reduces the network traffic, connection time and bandwidth consumption by the moving agent between the client and server. However, the security issue of the mobile agent makes difficult to acquire the benefits.

The HTML5-based mobile agent framework was developed in Tampere University of Technology (TUT). The core of this thesis is to secure the mobile agent framework. The security threats to the mobile agent and agent platform are classified to design and implement a secure framework. These threats are the agent attacking platform, platform attacking agent, agent attacking agent and agent system attacked by external entities.

This thesis focuses first two threats and provides a solution to protect mobile agent framework against them. The solution uses a signing method that involves salting and hashing of source address to generate signature. Furthermore, the RSA encryption using the static private key of an agent origin server to create a signature. The signature moves along with the agent and it is used to verify the agent source address using a static public key. This verification ensures that particular agent comes from the legitimate source and it is trusted as a non-malicious in the current platform. This solution overcomes the security issues like unauthorized access to the data, changing the agent and platform code, the misuse of others identity, eavesdropping and altering the important information, the excessive use of the resources etc. Also, the implementation helps to minimize the problems in agent mobility, agent and platform communication and identification of agents.

PREFACE

This Master's thesis has been written for the completion of M.Sc. Degree in Information Technology for the Department of Pervasive Computing at Tampere University of Technology (TUT), Tampere, Finland.

I would like thank my supervisor Prof. Kari Juhani Systä for recommending this topic and for the valuable advice, guidance and feedback. I am very grateful to my friends Sandeep Kumar Shrestha, Kishor Lamichhane, Bishwa Prasad Subedi, Veerendra Kumar Devarashetty for the support. Special thanks to Srijan Manandhar and Bikram Thapa for technical guidance.

Shiva Bhattarai
Tampere, 12^{th} March 2016.

CONTENTS

List of Figures

LISTINGS

1. INTRODUCTION

The HTML5-based mobile agent framework [1; 2] was developed in TUT. The framework transfers the application and its state i.e. inner data of an application, from the browser to server, server to the browser and server to another server [1]. The application in the framework is the mobile agent, that moves autonomously in various host [4]. The state of the mobile agent can be preserve, transport to the new host and continue its execution in new host [3].

It is believed that the mobile agent paradigm is a promising technology of the near future. It has a wide range of applications in electronic commerce, personal assistance, distributed information search and retrieval, monitoring, network management, real-time control, building middleware services, military command and control, parallel processing and the provider of important applications for the future mobile communication systems [13]. However, its mobility characteristics, as well as its ability to reside in different hosts, makes more vulnerable. So it requires some security services to preserve its originality. The security services should preserve an identification and mobility of an agent. It should guarantee the authenticity of an agent and platform in the framework. [5]. The purpose of this thesis is to design and implement a security framework. A proper security framework includes secured agent mobility, identification, and authentication of an agent [1].

The design and implementation described in this thesis use a signing method that creates a signature using the source address of an agent. In the framework, the agent is always downloaded from the agent origin server that is pointed by the source address. When an agent migrates, the source address is delivered along with the agent that is used to fetch the agent content. So, it is important to prevent the unnecessary modification of the source address. The signing method creates a signature to prevent the source address from the modification. The agent carries the signature and verifies in each host it moves.The proper verification will confirm that there is no modification in the source address of an agent. Furthermore, the receiver of an agent will ensure that the current agent comes from the legitimate origin server.

On the basis of an agent and platform i.e. an environment where an agent originates and executes, the security threats of the mobile agent technology are categorized into 4 types. They are the agent attacking platform, platform attacking agent, agent attacking agent and agent system attacked by other entities. However, the security implementation in the thesis gives solution for first two threats. In the solution, the security implementa-

tion works with the life cycle i.e., the agent stage from the origin to the exit of the mobile agent and its source address is authenticated in every host it moves. The authenticated source address guarantees that the particular agent comes from the trusted source. Also, an agent can be identified by its origin server. Furthermore, Chapter 2 presents the theoretical background of mobile agent technology. Chapter 3 presents the overview of the HTML5-based mobile agent framework. Chapter 4 categorizes the threats on the mobile agent component i.e., the agent and platform. It also provides the security measures for securing the agent and platform. Chapter 5 describes the security framework and its implementation method for this thesis. Chapter 6 evaluates the security implementation and it also presents drawback and future work for security implementation. Finally, Chapter 7 presents the conclusion of this thesis.

2. BACKGROUND

This chapter focuses on describing the basic concepts of agent, mobile agent, its advantages, their types and multi-agent system.

2.1 Introduction

The agent technology was originated from the Artificial intelligence(AI) and distributed computing research. The purpose of AI research is to use intelligent computing entities i.e. learning and adjust itself in an environment. On the other hand, the distributed computing will help to execute a task by cooperating several distributed agents on interconnected computers [7].

The term "agent" has different uses in different domains. So, there is no any universal definition that can be accepted everywhere. The dictionary meaning says that *an agent is a computing entity that performs some tasks on the behalf of somebody or something* [6]. For the context of this thesis, an agent is a computer program that is programmed to do specific work on behalf of others [6]. According to Bradshaw [8], *an agent is continuous and autonomous in nature*. An agent can move and function continuously in an environment over a long period of time and it is able to learn from its experience. Without any guidance, an agent is capable of doing its activities in a flexible and intelligent manner in a changing environment. Also, an agent can cooperate with other agents and processes in an environment [8]. In addition, the communication among the agents makes them an excellent candidate for distributed computing and software development. Thus, an agent can be used for the software development, such agent is called "software agent" [10]. It has general properties, they are as follows [11; 12]:

- **Reactive:** The ability to perceive the current context and changes of an environment and act in a timely fashion to the changes.
- **Autonomous:** An agent can control its action and internal state without any other support.
- **Goal-driven:** An agent has goal-directed behavior.
- **Temporally continuous:** An agent runs continuously deciding when to perform some activities.

There are some additional properties that help to distinguish an agent from the ordinary program. Such distinctive properties are communicative, social, mobility and learning. The communicative property of an agent helps to interact with other agents in the network. The mobility property refers that an agent can be moved from one environment to another to get enough resources for execution. Similarly, the learning property refers to the ability of an agent to learn from the past experience in an environment [12]. Furthermore, the additional property "mobility" classifies software agents into two types : stationary and mobile agents. The stationary agents are not able to move from the originated environment. They work and execute on their originated environment. On the other hand, the mobile agents can move between the various hosts in the network. This ability to move helps an agent to transport their state and code with it to another environment and resumes the execution. Here, the term "state" refers to the attribute properties that determine what and when to resume execution in a new environment. It is also called the execution state . Similarly the term "code" is object-oriented context, class code that is necessary for the agent to execute [12].

2.2 Components of mobile agent technology

Mobile agent technology consists of two components: mobile agent and mobile agent platform. This section describes the mobile agent and mobile agent platform.

2.2.1 Mobile agent

From Section 2.1, the mobile agent is defined as *the self-contained and identifiable computer programs that can move over the networks and act on behalf of the user or any other entity*. An agent can suspend execution in a current environment, transport its code and state to another environment and resume execution in that new environment. It is capable of detecting and adapting dynamically to changes in an environment [7]. The mobile agent is also called special type of mobile code [37]. The mobile code is such code that is sourced from remote, possibly untrusted systems and it is executed in the local system [24].

2.2.2 Mobile agent platform

A platform is a basic environment where compatible computer system and application programs can be developed and run, as a specific computer processor or network connection(hardware platform) or an operating system, database etc. The mobile agent is created and executed in a mobile agent platform. The platform where an agent is created is called "Home Platform" [17]. From the security point of view, it is the most trusted environment for that agent. The agent user defines an agent name and starts the agent from the home platform. An agent has its owner information, that is useful to prove itself

trustworthy to the other platforms. The owner information also helps an agent to return its home usually, the agent returns its home after the completion of the task [17; 37]. Furthermore, the platform controls the execution of the agents and provides other basic function such as agent communication, agent control, security, and migration [36]. Some of the existing examples of the mobile agent platforms are, Telescript, Concordia, Java Agent Development Framework (JADF), Voyager, Tromso And Cornell Moving Agents (TACOMA), Grasshopper, XML-based mobile agents etc.

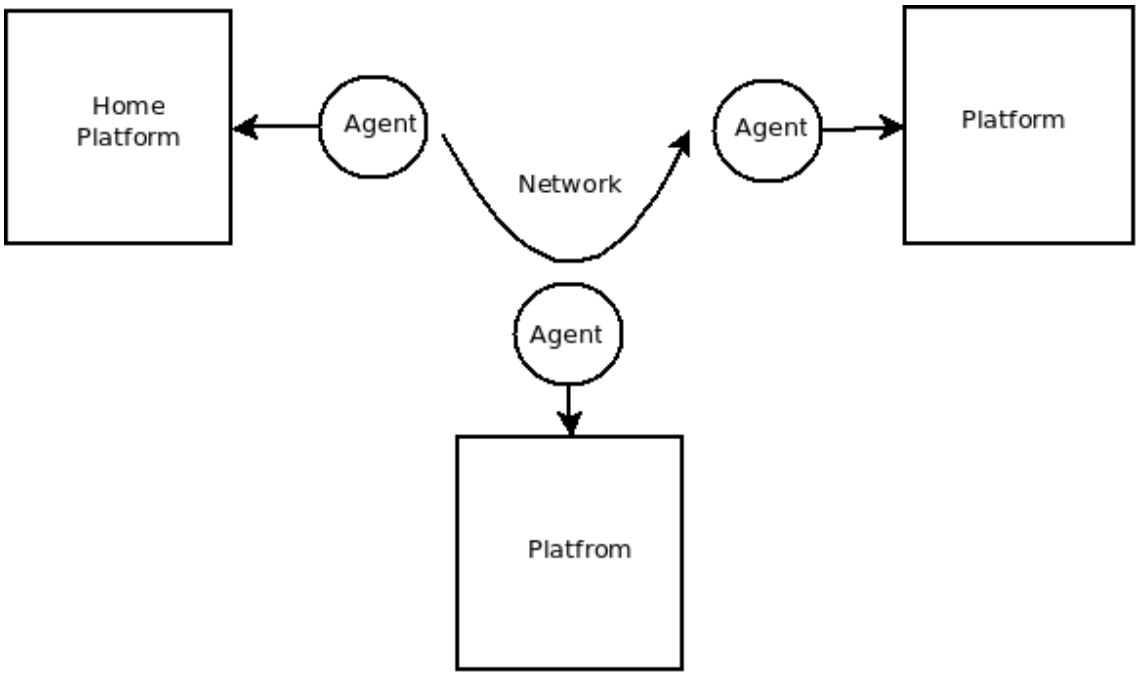

Figure 2.1: Mobile agent technology [17].

2.3 Why mobile agent?

The major objectives of the mobile agent are to reduce the network traffic and asynchronous interaction. The use of mobile agents will reduce the connection time and bandwidth consumption for processing the data between the server and client. This is possible only by moving the agents between the hosts. Similarly, the mobile agent also supports asynchronous computation. The agent can operate in an environment, without the continuous connection of the platform that has started. The mobile agent technology has been proposed as an alternative to a client-server paradigm. It is envisioned as a promising paradigm to deal with dynamic, heterogeneous and changing environments, that are suitable for modern Internet applications. The figure below shows the difference between the client- server and mobile agent-based paradigm [7].

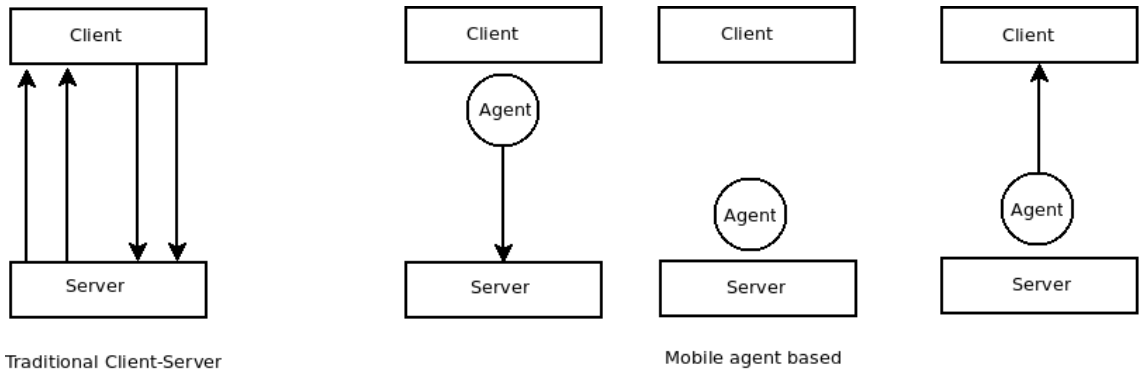

Figure 2.2: Client-server vs mobile agent based technology [7].

Other reasons are as follow [7; 12; 29]:

a. The mobile agent facilitates the system with high quality, high performance, economic applications. It uses all resources as a local resource, rather than accessing it remotely in other traditional paradigms.

b. It enables the use of portable, low-cost, personal communications devices. It contains a lightweight server that manages the movement of agents in the networks.

c. Efficient and economical uses of low bandwidth, high latency, error-prone communication channels. It has a mechanism of storing and forwarding of agents in the networks along with advanced queuing and persistent checkpoints.

d. Adapt and react autonomously in the changed environment.

e. The heterogeneous nature of the mobile agent will always have flexibility on heterogeneous the distributed system.

f. Mobile agents can be used in different applications such as diagnostic, e-commerce, entertainment, broadcasting, intrusion detection, home health care and many more.

2.4 Topology of agent

This section attempts to classify to study a topology of agents. The topology is the study of certain properties that do not change as geometric figure changes. Nwana [34] proposes four dimensions of agent topology [34; 35]:

- **Mobility:** As discussed in Section 2.1, one of the additional properties of the software agent is mobility. Based on the agent mobility in a network classifies them into two: stationary and mobile agent.

- **Behaviour:** On the basis of its behaviour, the agent can be classified into the deliberative agent and reactive agent. The deliberative agent is derived from the deliberative thinking paradigm. It possesses an internal symbolic, reasoning model and the engagement in planning and negotiation for the coordination with other agents. On the other hand, reactive agent identifies suitable environment, its changes and responds according to the current status of an environment.

- **Ideal and primary attributes:** This dimension includes three attributes: autonomy, cooperation, and learning. Autonomy refers that agents can operate without any guidance. An agent consists of individual internal states and goals that are maintained to achieve on the behalf of its users. The key element of autonomy is agent pro-activeness, i.e. ability to take an initiative. Similarly in a multi agent system, cooperation between multiple agents is a prerequisite. The agent possesses social ability to interact with other agents using communication language that helps in coordination as well. An agent can be smart with learning from their action and other from the environment. This ability of learning makes them intelligent agents. Using these attributes we can derive four agents in the topology: Collaborative agents, collaborative learning agents, interface agents and smart agents.

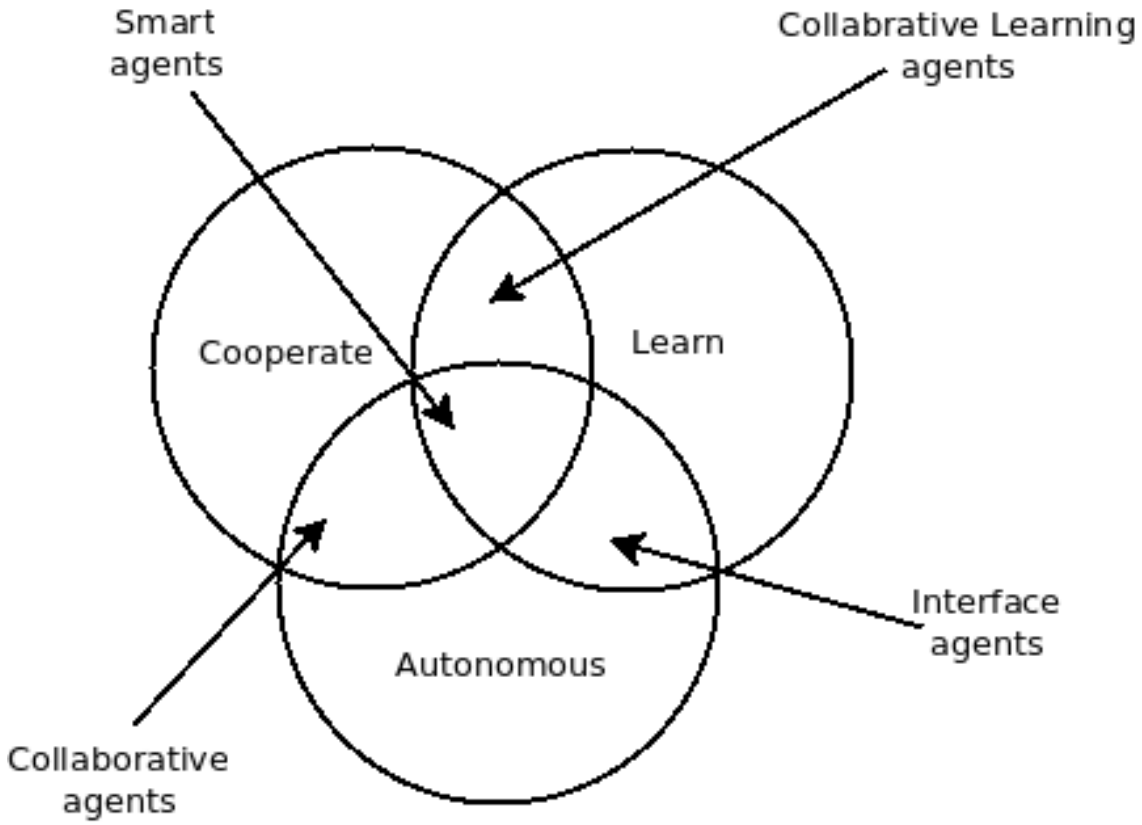

Figure 2.3: Nawana's Software agent topology [34].

- **Role:** With the role, information/ Internet agents are derived. These agents are the tool for managing the explosive growth of information. There are various roles to manage information such as managing, manipulating or collecting information from distributed sources. This agent will exploit Internet search engines and manages a large amount of information on the Internet.

2.5 Multiagent system (M.A.S.)

Multiagent system (M.A.S.) is based on agent technology. MAS consist of a computational system where multiple agents interact for their own objectives and good for overall systems. The multi agent system consists of good cooperation, co-ordination, and negotiation among abilities that helps in the completion of agent objectives. Cooperation refers to good communication among the agents. Coordination is about organizing various actions of the agent to achieve goals. Similarly, negotiation is about satisfying all parties of a system [9; 28].

A multi agent system is comprised of multiple autonomous agents and they have character like: [5]:

a. An agent depends on the system, sometimes they had to depend on other agents as well. So the agent cannot solve problems without a support of others.

b. No global controlling system and decentralized data, a multi agent system is envisioned as the system that communicate over the Internet, means anybody is able to connect with any platform and agents. So, MAS can be called a system with no global controlling system. Similarly, data in such systems are highly decentralized.

3. HTML5-BASED MOBILE AGENT FRAMEWORK

This chapter describes the idea behind HTML5 based mobile agent framework, architectural overview and execution. This chapter is mostly based on the given references of the thesis and papers [1; 2; 26; 38].

3.1 Introduction

The latest version of HTML family, HTML5 extends the applicability of the technology towards the client(-side) applications. It allows more application to be run in browsers and that application can be deployed over networks using the standard web technologies. The HTML5 mobile agent framework is based on HTML5 technologies. Here, the HTML5 agent runs in two modes. One with a user-interface inside browsers and other in a headless mode i.e., without a user interface that runs on an agent server. When the state of an agent is changed between the server and the browser, its state is saved. With the help of saved state of the particular agent will be able to continue its execution in a different environment, where it has been stopped before [1; 38].

The HTML5-based mobile agent framework uses a mobile agent paradigm to transfer an agent from the server to browser and vice-versa. It also uses the code-on-demand paradigm i.e., a paradigm where a code is downloaded to the client from different origin for execution, to get the static file of an application when an agent travels in the network. The browser represents the client side and agent server represents the server side of a mobile agent framework. An agent application is in the running state and continues its execution when it is on the server. Later on, the running application can be transferred to a browser. The HTML5 agent consists of two parts [1; 26]:

a. Implementation of the user interface is done with HTML5, CSS and image files.

b. JavaScript is used to implement the executable content and dynamic aspects of a user interface.

The reason for choosing HTML5 for implementation is that WEB is everywhere and the platform will be widely available. Also, the HTML5 controls user and makes an agent more user-oriented. The server side execution is done with Node.js. Thus both, the client and server are implemented with one common programming language. i.e. JavaScript [1].

3.2 Architecture of HTML5-based mobile agent framework

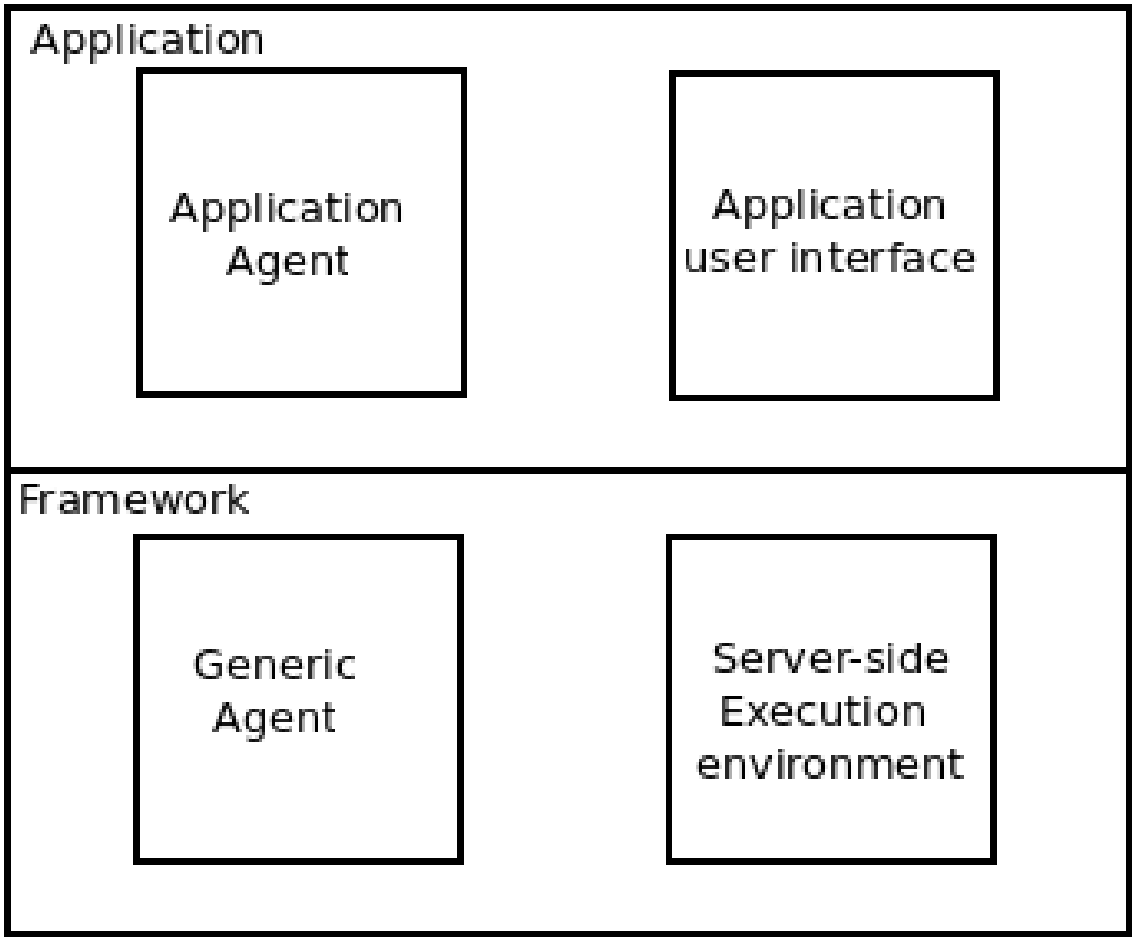

Figure 3.1: Basic architecture of HTML5-based mobile agent framework [1].

Above Figure 3.1 presents the basic architecture of HTML5- based mobile agent framework. The architecture is subdivided into two parts: application and framework. On the basis of the agent, it consists of two parts: a generic agent part and an application agent part. The generic interface provided by a generic agent that is used to access the server agent. The generic class is the base class for each agent applications and provides generic parts of an agent to all agent applications. On the other hand, the application agent part accesses the browser agent. It is the real implementation of this application which moves between browser and server [1].

3.2.1 Generic agent

It is the base class for agent application that provides generic services to agents and it is not instantiated. The generic agent is responsible for preserving the state of an agent. The basic structure of the generic agent is shown below:

```
function Agent(src, html) {
  var that = {};
        that.auri = src;
        that.huri = html;
        that.method1 = function(args) {.....}
        that.method2 = function(args) {.....}
        return that;
    }
```

Listing 3.1: Basic structure of generic agent [1]

Where, *that* is the variable where all the functions and variables also the application agent has access should be saved. The parameters *src* and *html* are the initialization parameter that are URLs to JavaScript and HTML file respectively. *auri* gives JavaScript and *huri* gives HTML based UI code file [1].

3.2.2 Application agent

It is the concrete implementation of the application that moves in between the browser and the server. An example of inheritance from the generic agent is shown below:

```
function MyAgent (src , html ) {
  var that = new Agent (src , html );
  /* new and overridden methods */
  return that ;
}
```

Listing 3.2: Inheriting from generic agent [1]

Where, *that* is a variable responsible for saving agent [1].

3.3 Execution of HTML5 based mobile agent

The agent life-cycle starts with downloading HTML agent from the origin server. The life-cycle refers to the various agent stage from the origin to exit. Similar to today's web applications, the HTML file of the agent includes references to Cascading Style Sheets (CSS), to other HTML files, images and other resources and JavaScript. The protocols are Web-friendly and rely on standard HTTP. Both origin server and agent server are HTTP servers that can be accessed with HTTP request i.e., GET and POST. GET is used to fetch agents to execute from the server while POST is used to push the agent to the server. This mechanism shows that agent can move from one server to another [2]. Some functions are discussed below, that are used in the framework.

- **/list:** This function list the active agents as an HTML file, which can be shown in browsers.

- **/upload:** This function sends URLs to agent code and the user interface together with the serialized state. The agent server starts the agent after receiving URLs.
- **/<id>:** This function pause the agents in server, serialize the state and send it to the requesting browsers.

The agents are serialized when they are about to move in between servers and browsers. When an agent is about to move from one to another location, its state is serialized into JavaScript Object Notation(JSON) based on the state variable. The serialization contains URLs for agent functionality i.e., the JavaScript file and HTML based UI. The URLs are denoted by *auri* and *huri*. Also, it consists of unique id for identification and other sets of variables. An example of serialized agent content is shown below [2]:

```
{"auri":"http://localhost:8890/ClockExample.js",
    "huri":"http://localhost:8890/ClockExample.html",
    "id":"526636",
    "memory": {
      "high":"0.053",
      "low":"0.0214",
      "count":"3",
       "history":[0.0253,0.0234,0.0214]
    }
}
```

Listing 3.3: An example of serialized agent [2]

The code is often downloaded from the origin server pointed by *auri* and *huri*. The *auri* gives JavaScript code and *huri* gives HTML based UI code of the serialized agent. When the agent server receives a serialized agent description i.e. JSON file, it fetches the JavaScript code from *auri*. Additionally, an agent downloads other JavaScript files for agent implementation that helps in initializing the agent and starts the execution of the agent. When a browser requests the agent from the agent server, it responds to the browser by sending the content of HTML file identified by the *huri* field of agent description. The agent server injects JavaScript to HTML file so that agent can be resumed in the local environment. [2]. The life cycle of an HTML5 agent in the framework is shown below:

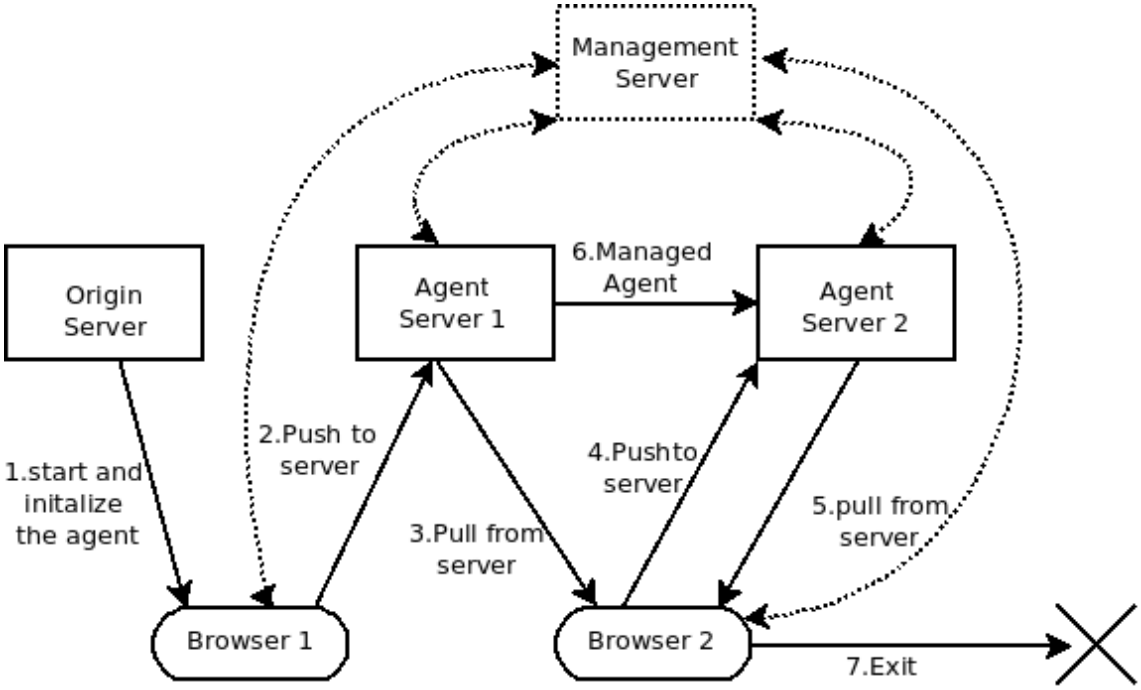

Figure 3.2: Life-cycle of an HTML5 agent in framework [2].

4. SECURITY IN MOBILE AGENT TECHNOLOGY

This chapter focuses on describing the security threats in mobile agent technology. It also provides the survey of the techniques that can solve the problems of threats.

4.1 Introduction

As discussed above in Chapter 2, mobile agent technology is a useful approach for the distributed systems. It also has advantages over existing the client-server paradigm and it can be used in numerous applications on the Internet. However, there are various technical challenges in the use of mobile agents. Among of them the security is the primary issue in mobile agent technology. From the above chapters, we came to know that an agent can move autonomously and widely distributed in the network to accomplish the various task on the behalf of users. The autonomous characteristics of an agent will expose and make them vulnerable to various attack in the network [13].

There are various attacks in the mobile agent technology some are as follows: [13]:

- **Masquerading:** This attack tries to misuse the identity of others. The malicious mobile agent or platform claims the identity of others in order to get the access of other component resources.
- **Denial of services (DOS):** The malicious platform ignores the mobile agents requesting the services and resources from the platform. It terminates the execution of an agent without any notification or assign it continuously so, that it will never reach the goal. Similarly, the malicious agent consumes enormous amount of resources from the platform or deletes the important file. The other agent will be scarce of the resource, thus it causes harm to the platform and other mobile agents.
- **Annoyance attacks:** This opens many windows on the platform or tries to make the computer beep repeatedly.
- **Eavesdropping:** The malicious platform monitors the behavior of the mobile agent to get the important information from the mobile agent. Monitoring can be used for identifying communication channels or can be other requests to the mobile agent.
- **Alteration:** The modification of mobile agent information by insertion, deletion or altering the agent code, data and execution state. The modification of code or the execution state causes malfunctioning in platforms and agents.

The characteristics of the secure mobile agent paradigm are as follows [1][4]:

- **Confidentiality:** The information of a mobile agent and stored in the platform should only be accessible for the authorized parties.
- **Integrity:** The protection of mobile agent code, the state and the data from being modified.Detecting and preventing the unauthorized modification will help to achieve this characteristic.
- **Availability:** The mobile agent has huge demands of services and data from the platform. If requirements are not met, then platform needs to notify. Also, the platform should afford a certain level of fault tolerance and fault recovery from unpredicted and hardware failure.
- **Accountability:** The audit logs are maintained to keep the tracks of mobile agent actions. The audit log will help to recover important files in a situation of system failure.
- **Anonymity:** The actions of the mobile agent in audit logs should be private and the platform needs to balance the need for audit logs.
- **Authentication:** The process of proving one's identity.
- **Non-repudiation:** The mechanism to prove that sender has really sent this message. The sender will not be able to deny about the message that it has sent.

Furthermore, a secure mobile agent system develops a trusted environment. Unfortunately, there is no such explicit definition of trust. According to Oxford dictionary, *It is a belief that somebody or something is good, sincere, honest etc. and will not try to harm or trick you*. The security mechanism should maintain the trust among the entities of the mobile agent paradigm. The trusted component of the mobile agent does not harm other component and networks. For example, a home platform is always trusted environment for an agent [5; 31]. The trust in a component depends upon the reputation. In other words creating trust between entities is possible with its history and knowledge of each other. In mobile agent technology, the sources of components can be either trusted or untrusted. In trusted scenarios, it is assumed that there will not be the situation of threat to any mobile agent entities. The threat and other security issue are only from the untrusted sources. So, to trust and use the resources of untrusted sources, they are authenticated. Authentication is necessary to make sure that the resource of untrusted sources will not have any negative effect on the mobile agent system [5; 31].

4.2 Security threats in mobile agent technology

As discussed above in Chapter 2, mobile agent technology has two components: an agent and agent platform. The first component is responsible for the code and state information. The latter component is responsible for providing the computational environment for the agents. Based on the agent and platform, the security threats are categorized into four: the agent attacking platform threat, the platform attacking agent threat, the malicious agents attacking agent threat, the agent system attacked by other entities [14].

4.2.1 Agent attacking platform/host threat

The agent attacking the platform refers to the threat of launching attack against the platform by exploiting its weakness. The malicious agent can have some ways to attack on the platform to get unauthorized access to information, masquerading, the denial of service attack etc. Unauthorized access is a threat that occurs in the platform due to lack of adequate access control mechanism. The unauthorized agent in the platform claims themselves to be a trusted agent to gain the access to services and the resources of the platform. Behaving as trusted agent situation is called masquerading. A masquerading agent destroys the trust and reputation which was developed by the legal agents in the community of the platform and networks. A masquerading agent gains access, information, confidential data, instruction and code related to the platform. With the access level these agents tries to tamper and render the resources by modifying information or creates the situation of Denial of services. The agent in Denial of service (DOS) attack, consumes excessive amount of resources and services, and denying to share the resources. The constant consumption of resources and the services, also blocking other agent by overloading will creates deadlock [14; 15; 17].

In addition, there can be the complex form of attack such as the event triggered attacks or logic bomb. When code, concealed within a peaceful mobile agent which is triggered by specific events such as the time location or arrival of specific person example the trojan horse program [14; 19]. Furthermore, in some self modifiable programming language like JavaScript, the code modifies itself. When, the agent code modifies itself it is impossible to detect and know what the agent will be doing when it is executed in the platform. This self-modifiable characteristics can be very harmful for the platform in the mobile agent paradigm [2].

4.2.2 Platform/host attacking agent threat

The platform attacking agents refers to the threats of the malicious platform to agents. The threats can be masquerading, denial of services, eavesdropping and alteration. An agent consists of sensitive information such as collected data, information log that agent

has visited. So a malicious platform can easily capture and attack an agent by extracting information, corrupting and modification of information, code and information log. It can also deny the requested services or re-initialize or terminate completely. Similarly eavesdropping and spying threats causes the interception, finding the secret security channels. These effects on the agent behavior and accuracy of computation so, the outcome of an agent can be wrong. The malicious platform may behave as a legitimate platform and try to convince the legal agent. Similarly, a platform may not accept the request from the agent and does not provide necessary resources which will create a situation of denial of service attack [14; 15; 17].

4.2.3 Malicious agent attacking other agent threat

The malicious agent attacking the agent refers to the threats of malicious agents that exploit the weakness of legal agents or launching attack against other agents. The threats can be masquerading, unauthorized accessing, denial of services, masking, repudiation. It can also have a threat of false transactions due to eavesdropping or interference with agent activities. An agent can get information and authorization by serving as an intermediary to the target agent through masquerading. The problem of the malicious agent attacking the agent also occurs due to poor access control mechanism in the platform. An unauthorized agent gets access to the platform and modifies the information about the agent and code or interfere by using it public methods. Interference refers to the masking of a legitimate agent with the malicious agent. It also collects the enormous amount of resources and do not share with others, it leads to the deadlock. Also repeatedly sending messages or spamming to another agent creates a denial of services attack situation where the receiver cannot handle the messages. This will lead to the agent crash. Similarly repudiation threat, where an agent participates in transactions or communication but always claims that there is no transaction or communication [14; 17].

4.2.4 Agent system attack by other entities

The agent system can be harmed in both outside and inside of the agent framework. The attack of other entities to the agent system refers to the external entities, including agents and agent platforms. The threats can be masquerading, denial of services, unauthorized access, copy and the reply. The entities are capable of intercepting the agent or messages in transit and modifying the content, mask or replace other contents. Since agent services offered by the platform and inter platform communication can have a threat of DOS attack. The request from the remote users, processes and agents that are not legal can get access in the mobile agent system. Due to the autonomous moving nature of the agent, the third entities that intercept an agent or agent message can attempt to copy or clone agent and agent messages [17].

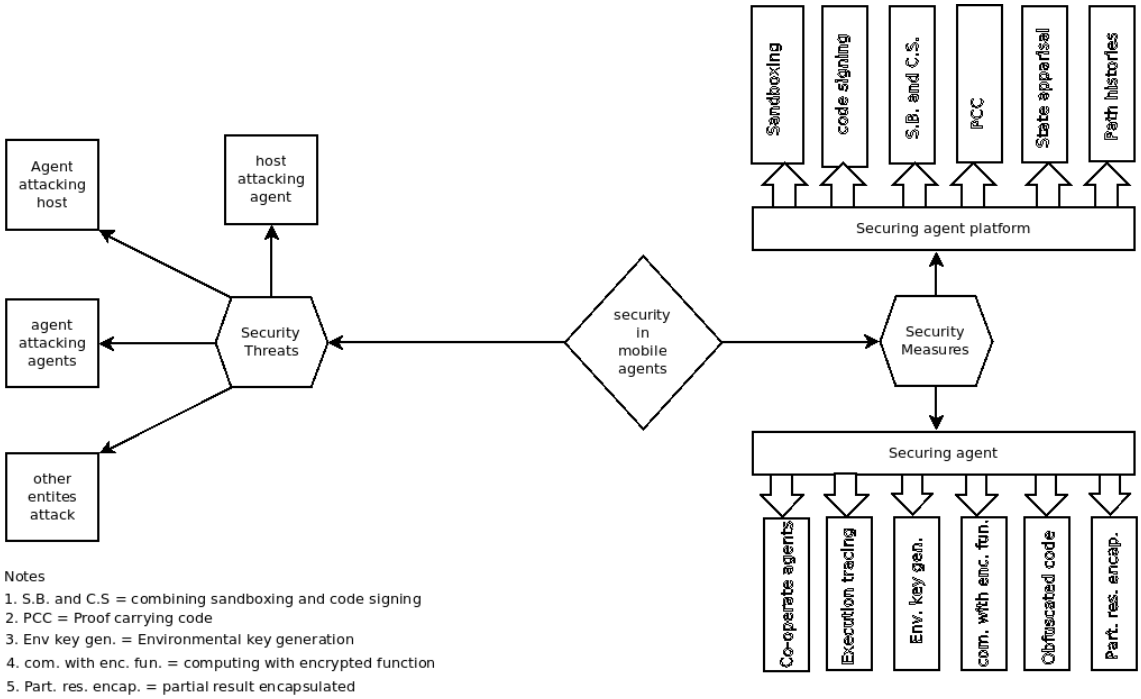

Figure 4.1: Summary of security in mobile agent

4.3 Security measures in mobile agent technology

Security measures are different actions, devices, techniques, procedures, methods or any measures that try to prevent the vulnerability of a threat in the mobile agent system. Since, the various security measures have been applied to protect the entities of mobile agent systems depending upon the mobile agent paradigm and feasibility. The security measures in mobile agent systems needs some common set of baseline assumption based on trust [14].

a. The agent should trust the origin platform and start execution.

b. The agent origin platform/origin server and other agent servers should be equally trusted, implement security and should not behave malicious.

This thesis focuses on securing the agent and agent platforms :

4.3.1 Securing the agent platform

Securing the agent platform is major issue in the securing mobile agent system. It is susceptible to attack launched by mobile agents and other various entities. This chapter presents various the detection and prevention method that tries to maintain security in the agent platform [13; 14].

- **Sandboxing:** It is also called Software-Based Fault Isolation [14]. This method uses software technique that separate local or trusted code from untrusted or remote code. Trusted code is executed with full permission and can access the resources. The untrusted code such as the mobile agent and the downloadable applet will be executed in a restricted area called "sandbox" [13]. The sandbox applies some security policy for the execution of remote code. This policy controls the access of system resources, opening the new network connection and invoke programs on the current system [13; 14].

  Sandboxing mechanism is common in Java interpreter inside Java enabled the web platform. It contains three security level [13].

  a. ClassLoader: It is responsible for converting remote code into data structure which is added to the local class hierarchy.

  b. Verifier: The verifier is responsible for security checking before loading remote code and remote code should valid virtual machine code.

  c. Security Manager: The security manager is responsible for checking the remote class and grant access of resources using verification.

  Problem

  a. Three parts of Java interpreter inside Java enabled the web platform are interrelated, so the failure of any of them leads to a security violations [13].

  b. This method increases the execution time of legal remote code [13].

- **Code Signing:** Code signing technique involves public key cryptography to ensure the integrity of remote code. The public key cryptography uses key pair, the private and the public key. The private key is kept secret while a public key is distributed freely. From the pair, one key is for encryption i.e., the process of transforming readable information to unreadable or unintelligible form. Similarly other key is used to decryption i.e. transforming encrypted information to the original state [25]. The code signing uses the digital signature that employs the public key cryptography and hash algorithm. The hash algorithm is responsible for encrypting the original messages into fixed length hash value. The hash value is also called the message digest. The hash value is very difficult to recover in the original state. For example SHA-256. SHA-256 comes in the family of SHA-2 and stands for " Secure hash algorithm" designed by National security agency (NSA) of the USA [43]. The digital signature may use the salt because of its randomly generating property. Salting is a technique in cryptography, that uses randomly generated data as an additional input in the hash algorithm along with the original message. The

primary function of this technique is to defend against the “dictionary attack”. It is attack based on trying all the strings as possible [44; 45].

A digital signature is serves as handwritten signature but more secure than handwritten. It starts with calculating the hash value of the original message that is intended to send to other parties. Using the public key cryptographic techniques, hash value is signed with the private key to produce the digital signature. Thus the digital signature and the original message is send to the intended receiver. On the receiver side, the digital signature is decrypted to produce hash value with sender’s public key that has been distributed. Similarly, the hash value of the original message is also calculated. Finally, both hash values are compared to validate, the message sent by the intended sender in the network. Only identical hash value is validated. The working of the digital signature also depicted in the Figure 4.2 [16; 18; 25]:

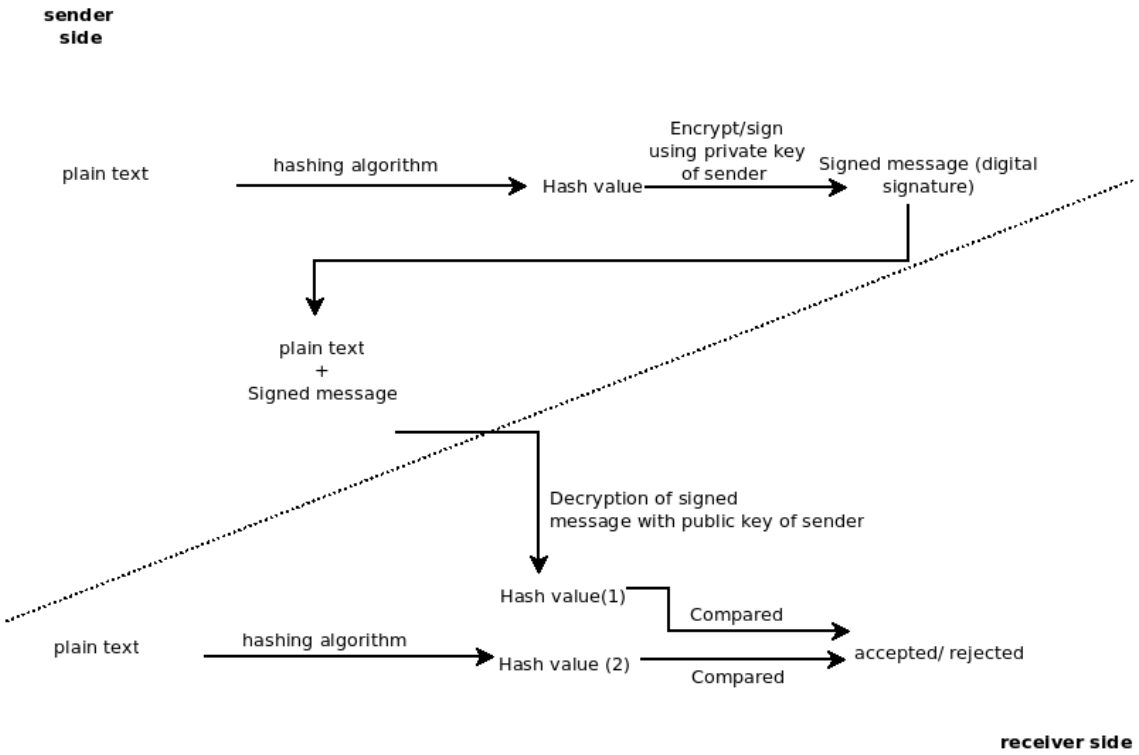

Figure 4.2: Digital signature and signature verification

The code signing techniques uses the digital signature and the hash algorithm to confirm the authenticity, originality and integrity of remote code. It also verifies the code that has not been corrupted or modified after signing, normally code is signed by its producer, user or some third party. For example, Microsoft Authenticode, it uses , the platform signing code such as ActiveX controls and Java applet [13; 14]. Code signing technique verifies the code creator and sometimes guarantees the code creator are trustworthy. It depends upon how code signing is used. So, in that case

to maintain trust, the platform where mobile code run maintain an index list of trusted source. So before running the code, the platform maps and checks the index of trusted source with the remote code source. Code is assumed to be trustworthy, if the remote code source is found in the index list of trusted source maintained by the platform. Finally, remote code is treated and given full privilege as trusted local code. Code signing treat code in trusted or untrusted form. This policy is known as "black-and-white" policy [13].

Problems

a. This approach assumes that the trust lists of the source maintained by the platform are always trustworthy and uncorrupted. If any malicious agent enters the platform,it not only affect the platform but also changes the policy of the platform, it opens the door for other malicious agents to enter. This attack effects will be seen later on that makes impossible to make connection between attack and attackers. So it is also called "delay attack" problem [13].

b. Sometimes code signing mechanism may not run the remote code from untrusted source, so this approach seems to be more restricted. It also depends upon how code signing is implemented [13].

- **Combining Sanboxing and Codesigning approach:** This approach tries to overcome the execution time increment of legal remote code and the over restriction over the legal agent from untrusted sources. For example, in JDK 1.1 the remote code is granted full privilege like local code when the signer of remote code is trusted. While a code is not trusted, it will runs into a security policy of sandbox. This approach combines the advantages of sandboxing and code signing that will facilitates the execution of the untrusted remote code. Still, it has the problem of "delay attack" [13] as in code signing approach [13].

  Similarly, in JDK 1.2 (Java 2) the advantages of both approaches are combined and it also incorporates access control and "shades-of-gray" policy [19]. Since "black and white" policy separates code into trusted and untrusted so that only trusted can be used. While "shades-of-gray" policy come up with the degree of partial trust in code. In JDK 1.2, local code and remote code are treated with the same security policy. At run-time, code is parted into groups called "protection domains" [13] and all code in particular group is granted with the same set of privileges. Also, the end-users can authorize certain "protection domains" to utilize maximum amount of resources. Additionally, in each domains different subsets of permission and privileges are granted [13; 19].

  Problems

JDK 1.2 combine the sandbox and code signing with shades-of-gray works well. Still, the malicious code can be granted with full privileges that create the similar problem of code signing approach and threaten the platform [13].

- **Proof carrying code (PCC):** The technique of Proof-Carrying Code (PCC) was introduced by Lee and Necula [13]. In the PCC method, the code creator provides formal proof of the code for the consumer with the security policy of the agent platform. This technique tries to prevent the execution of unsafe code. The safety proof and code are sent to the consumer, that compiles with security policy and verified. This is also called "machine checkable proof" [13]. Thus, with the result from the validation process, code is executed if it is safe else rejected. PCC is also called "self-certifying" [1] , since not cryptographic or any third party authentication is required. It is the low-cost static program checking without any expensive run-time. Using PCC, modification in code can be identified by reading the proof. So PCC is used as "tamper-proof" [13].

  Problems

  a. It has problems with the proof generation. Many of the proof are not automated, it needs lots of studies and research on automated proof generation [13].

  b. The time consumption of proof validation and size of proof are the problem [13].

- **State Appraisal:** According to W. M. Farmer "State Appraisal" [20] ensures that mobile agent has not been harmful or altered with the change in its platform or state. Since a moving agent consist of code, static data, collected data and the execution state. The state appraisal methodology has an appraisal function, that is owned by the both creator and user of the agent. The function becomes a part of agent code and agent is digitally signed by the both creator and the user. The appraisal function is now protected with undetectable modification. When an agent comes to a new environment, the agent platform uses the function to verify the correct state of the agent. In addition, it determines the privileges given to the agent during execution. On the basis of the result of appraisal function the platform security policy, privileges and permission are given to that agent [13; 14].

  The success depends upon to which extent unnecessary modification can be predicted and protect them from malicious entities. Here appraisal function verifies the state of agents and calculates the privilege for an agent depending upon the platform policy. The signed state should not be violated to get the privilege in the platform, otherwise the agent does not get any privilege. The advantage of this technique is that it is s flexible way to request privilege being on a particular platform

[13; 14].

Problems

a. It is difficult to obtain the state appraisal function [13].

b. It is not easy to formulate appropriate security polices [13].

- **Path Histories:** The purpose of path histories is to maintain a visited record of the agent in the different platform. Current platform needs information on the visited platform histories and to what extent an agent has transformed into malicious. Depending upon the information, an agent is trusted and the service, the resource and the permission is granted from the platform. Similarly, the path of the visited platform is recorded and signed by a current platform [13].

  Problems

  a. Path histories technique has the problems of the cost path which is visited by agents [13].

4.3.2 Securing the agent

Securing mobile agent technology also includes another part as major issue, securing the agents. It refers to securing an agent from the malicious platform. Agents are always susceptible to various threats of the agent platform that they visit, since agents are exposed in the mobile agent platform. The threats or malicious behavior cannot be prevented but it can be detected. This chapter presents various threat detection methods that will help to secure agents from maliciously behaving the platform [14].

- **Co-operating agents:** This technique uses a peer agent to protect themselves from a malicious platform. The co-operating agents of disjoint set of the platform share the same data and exchange information in a secret way. The agent, when moves between the platform, with the help of authenticated channel information are passed to its co-operating agents. Information includes the current platform, the state and the successor platform to be visited. Both co-operating agents will execute the similar task and preserve the information of it. The co-operating agent will take necessary action if anything goes wrong in the mobile agent. This also helps to reduce the modification of shared data. Co-operating agents can be for the e-commerce task or protocol such as the authorization of negotiation, bidding, auction electronic payment [13].

  Problems

  a. The high cost of setting up the authentication communication channels [13].

b. The co-operating agent may be killed or lost [13].

- **Execution Tracing:** Execution tracing involves cryptographic traces of mobile agents and its behavior in various the platform. The tracing agent behavior will help to get information about the unauthorized modification, state and execution flow. The execution traces are logs of each agent performance and history of the different platform. The traces consist of statement identifiers and the signature of platform activities performed in a particular platform. For verification the statement in traces can uses agent values of internal variable are called “white” [13] statements. However, the statement that requires external information so trace must contain the digital signature of the platform and requires third party authentication. This statement is called “black” [13] statement. Execution tracing assumes that all parties such as agent user, platform uses public and private key and they communicate through cryptographic signed messages. Traces are used to verify and protect the legitimate agent and the performance from malicious entities [13; 14].

  Problems

  a. Verification is not compulsory until any suspicious condition occurs. However, Tan and Moreau [13] has suggested a new technique to have compulsory verification [13].

  b. Tracing every movement of agent yields large size and number of logs, this creates managing problems [13].

  c. Verification servers can have collaboration with a platform and servers [13].

- **Environmental key generation:** Environmental key generation technique requires some environmental condition to be true to access the mobile agent. The environmental condition can be matching strings. When a platform wants to communicate with another platform, an agent is send with encrypted information with it. The encrypted information may contain some data or some executable codes. The agent itself cannot predict and visualize the information. So, the agent itself is clueless and waits for the environmental condition to met. Hence, the agent is also called "clueless agent" [27]. The agent will go through the process of meeting environmental condition in the receiving platform. If condition happens to be true, the information generates a key that decrypts the information carried by the agent. The agent is not able to decrypt its own message without meeting the environmental conditions. The technique of generating key will ensures that platform or any entities cannot affect the agents only by reading the agent’s code [13; 14; 27].

  Problems

  a. The receiver platform can change after environment key generation [13] .

b. The platform can be considered unsafe to execute the code that is attached with the mobile agent [13].

- **Non-Interactive Computing with Encrypted function:** Computing with encrypted function is a software solution, that is responsible for determining a methodology for the safe computation of cryptographic primitives, such as the digital signature. This technique uses an encrypted function along with the executable program on a mobile agent platform [13; 14]. Non-interactive computing involves the owner of an agent to execute encrypted programs over the untrusted platform. For example: The home platform has an algorithm to compute a function f. The target platform has an input x is able to provide services to the home platform by computing f(x). However, the home platform does not wants the target platform to learn anything about function f. So, the home platform encrypts the function f and gets E(f). The home platform creates a program P(E(f)) that uses E(f). The P(E(f)) is send along with the mobile agent to the target platform for execution. The agent is received at the target platform and execute it. The execution also includes program P(E(f)) at x to produce P(E(f))(x). The agent is send back to its home platform. Now the home platform extracts the result from the agent and decrypts to get f(x) [1]. [13].

  Problems

  a. Finding a way to apply it to an arbitrary function f [13].

  b. It is still vulnerable to certain attack such as denial of services and replay attacks [13].

- **Obfuscated Code:** Hohl [13; 14], gives obfuscation technique to use the blackbox security. The blackbox security is the technique of scramble code, so that no one is able to understand the code. It is difficult to modify the code without understanding. Obfuscated code technique tries to prevent the mobile code from being analyzed or read by the host. It involves the mobile code modification and preserve the semantic of code such that the host is not able to learn or understand them. There are some obfuscated code types such as layout obfuscation, remove or modify some information in the code such as comments and debugging information. Data obfuscation is responsible for modifying the data and the structure of data without harming the execution. Control obfuscation modifies the control in the code. Similarly preventive Obfuscation protects code from decapitators and debuggers [13; 14].

  Problems

  a. There is no known algorithm or any approach for providing the blackbox protection [13].

- **Partial Result Encapsulation:** Partial Result Encapsulation (PRE) is used for finding the security breaches of agents when it executes in the different platform. It encapsulates the output of each agent performances at the particular platform. These encapsulated information uses for verification to ensure that agents are not affected by the malicious platform. The verification can be in a home platform or in some intermediate place.The purpose of an agent performance encapsulation provides integrity, accountability, confidentiality and privacy. Depending upon our goal, information is encapsulated in the platform. For example, certain security requirement such as integrity, accountability and privacy of agent uses cryptographic primitives such as encryption, digital signature, authentication codes and hash functions. For confidential, the encrypted result is decrypted with the public key [13].

  Furthermore, Young and Yung propose "Sliding Encryption" [13]. It encrypts small amount of data from large amount. Another method is "Partial Result Authentication Code" (PARC) [13], the creator of the agent maintains the list of secret key. The secret key is supplied to the agent for PARC computation. The key is used once with encapsulated information and destroyed before migrating to the next platform and guarantee "forward integrity" [13]. It enables that there is no change in the platform that are visited in future by the previously visited platform, since there is no secret key to compute PARC. The problem with PARC is that malicious platform can generate copies of original keys. The revisit of the agent will causes modification. PARC focuses on integrity but no confidentiality is maintained. Karjoth [13], improves the PARC technique and come up with "Strong forward integrity" [13] where the visited platform cannot be modifies its own previous result. In this process, encapsulation is done by the platform instead of the agent. It uses creator's public key to encrypt the agent result, thus confidentiality is maintained. The encrypted result is signed with the private key and hash chain by the visited platform. The hash chain connects the outcome of the previous platform with the identification of the latter platform. This is how, the result is not modified later in revisit of the agent in the same platform [13].

  Problems

  a. PARC has the problem of changing the result of the platform which is revisited by the agent [13].

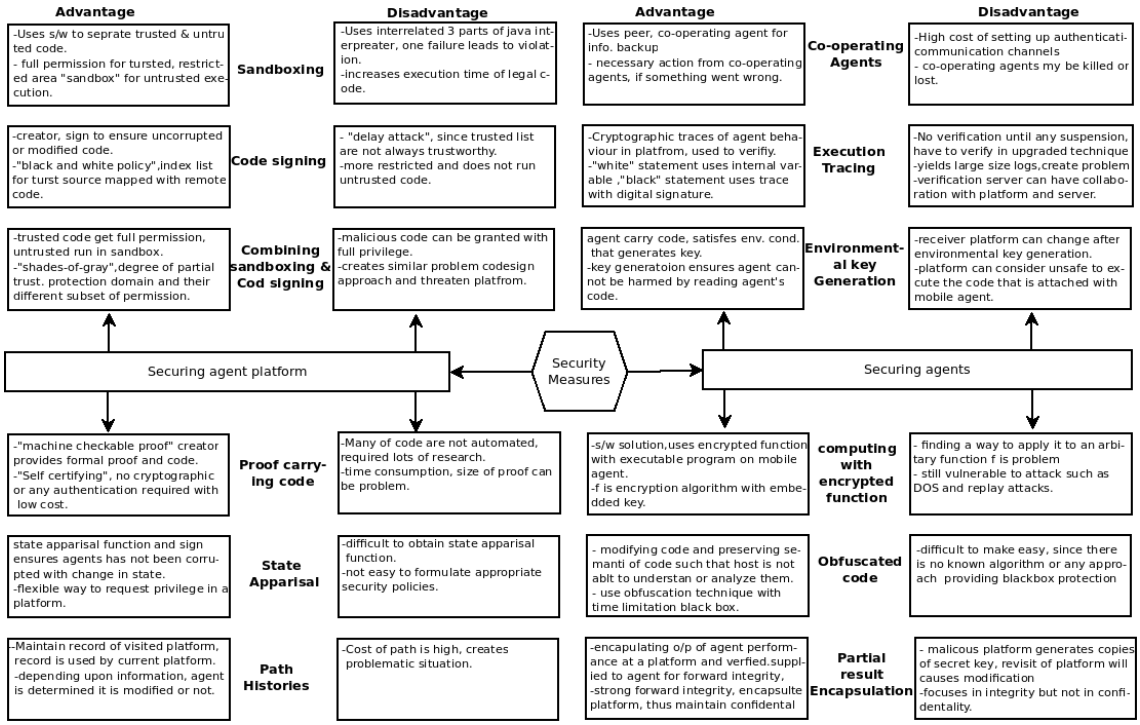

Figure 4.3: Summary of security measures

5. SECURITY IMPLEMENTATION ON HTML5-BASED MOBILE AGENT FRAMEWORK

This chapter describes the security solution for HTML5- based mobile agent framework. As discussed in previous security chapters a secure mobile agent framework should ensure that its agent and platform are not malicious. Also, the agents of the framework should provide authentic information and protect that information.

The motive of the security implementation is to develop a secure mobile agent framework that includes a secure agent mobility, identification and authentication of an agent. This security implementation mainly focuses on securing the source address of the mobile agent i.e., the source URL. A source URL is an address of the platform (also called origin platform/server) where an agent is created. So, it is an important part of the framework that is denoted by *auri* [2].

As discussed in the previous chapter, the instance of an agent is created in the origin platform and downloaded to a browser. The downloaded agent can move between different servers (also called the agent server) and browsers (the client of the network). When an agent moves, the source URL that points to the origin server is delivered to the agent receiving platform which is used to fetch the agent content [2]. The malicious entities can attack and modify the source URL that points to the malicious servers. The malicious server may provide a malicious agent that will affect on the platform by misusing the identity of the legitimate agent, consuming the enormous amount of the platform, eavesdrop on the platform, inserting and deleting the important data and code of the agent and platform etc. Hence, it is necessary to prevent the modification of a source URL and ensure that it points to the trusted origin server. However, this security solution is not enough for the whole framework, but it can be the major part of the security. As it protects an integral part of an agent framework.

5.1 Element used in security implementation

This section describes the elements used in security implementation of mobile agent framework.

5.1.1 Key pair and public key encryption

The security implementation uses an RSA key pair i.e., the private key and public key. The keys in a pair have a relationship and they are used for public key encryption i.e., encryption and decryption. In the implementation, the key pair is static so, they are called static private and static public key. Here the static means the keys are fixed, in the implementation keys will remain same. In comparison with the non-static keys i.e., frequently changing keys, the static keys are simple and easier to implement. The non-static keys are unique and generated by the trusted authority in the public key encryption. [49]. Whereas, a static private key is kept secret in agent origin server which encrypts the message in the public key encryption.

In the implementation hash value, i.e. a value produced by a hash algorithm using source URL is encrypted to create a digital signature. The signature is sent to the other parties in the framework. Sometimes, the malicious entity may discover the signature and use it to claim malicious source as a trusted source. The agent origin server always runs the encryption however, the malicious source may not run the encryption and used the previous signature to claim itself as trusted source.

The encryption process produces the same signature for a particular message using a static private key. This is the consequence of the static private key, that can be solved by using salt. The hash algorithm, digital signature and salt are the elements used in this thesis implementation that will be explained in next Section 5.1.2. On the other hand, the public key is in the framework that is used for the decryption of the messages. The signature is only decrypted with the static public key. It helps to ensure the authenticity of the source platform. It also provides a confidentiality that the information coming from a particular origin platform is legitimate.

For the purpose of this thesis, the key pair is generated from the RSA library of Herbert Hanewinkel [41]. The key generation is not part of this implementation as once the keys are generated and used as static private and public key. In general the RSA key pair is used in two ways [40]:

a. The public key is used for encryption and the private key is used for decryption. The message encrypted by the public key is only decrypted with the private key which is the other pair of the public key.

b. A message can be “signed” with the private key. The signed message can be verified by only with the public key, which is the other pair of the private key that signed the message.

Here, the signing means applying the decryption over the original message [40].

However, for this thesis purpose the key pairs are used differently, as the private key is used for encryption and the public key is used for decryption. Mathematically, in RSA

algorithm, encryption with private and decryption with the public key is also correct. Typically, encryption with the private key is called “sign” and decryption with the public key is called “verify”. The mathematical example is shown below [40; 30]:

The RSA algorithm starts with generating public and private key.

a. Selecting two random prime numbers, p and q. Let p = 5 and q = 11.

b. Computing n, as product of two prime numbers p and p. $n = pq$ = 55.

c. Calculating $\phi(n)$ as $\phi(n) = (p\text{-}1)\ (q\text{-}1) = 40$.

   Where, $\phi(n)$ is the Euler totient function giving number of positive integers less than n which are relatively prime to n. Two numbers are "relatively prime" when they have no common factors other than 1 [50].

   Now, selecting public exponent e, which is relative to $\phi(n)$ $1 < e < \phi(n)$

   i.e., $1 < e < 40$,

   it gives 3, 7, 9, 11, 13 ....

   Selecting e = 3 from the above list and it is the public exponent of public key.

d. Computing the private exponent d from e, p and q. It will be the private exponent of the private key.

   Finding the modular inverse of e with respect to $\phi(n)$

   i.e., $e * d \bmod \phi(n) = 1$.

   solving for d it gives d = 7.

Now,
Private key:

$$(n, d) = (55, 7) \tag{5.1}$$

Public key :

$$(n, e) = (55, 3) \tag{5.2}$$

Message :

$$m = 20 \tag{5.3}$$

The RSA encryption operation is exponentiation to e^{th} modulo n:

$$c = Encrypt(m) = m^e \ mod \ n \tag{5.4}$$

The RSA decryption operation is exponentiation to the d^{th} modulo n:

$$m = Decrypt(c) = c^d \ mod \ n \tag{5.5}$$

Where,

m = message

c = encrypted message

e = public exponent

d = private exponent

mod = modulo (%). It is the remainder when one integer is divided by another [51].

a. Encryption with public key and decryption with private key

$$c = Encrypt(20) = 20^3 \ mod \ 55 = 25$$
$$m = Decrypt(25) = 25^7 \ mod \ 55 = 20$$

b. Encryption with private key and decryption with public key

$$c = Encrypt(20) = 20^7 \ mod \ 55 = 15$$
$$m = Decrypt(15) = 15^7 \ mod \ 55 = 20$$

Furthermore, the keys in the implementation consist of constants and exponents. When keys are used for encryption and decryption, the constants and exponents is calculated. The exponents are the original key that is used for encryption and decryption.

5.1.2 Digital signature

The security implementation employs RSA algorithm, public-key encryption, hash algorithm and salt to create a digital signature. The general description about the digital signature, RSA algorithm, public key encryption, hash algorithm and salt is given in Section 4.3.1. The implementation uses "SHA-256" [43] as the hashing algorithm that is used over the source URL of an agent origin server to produce the hash value. As discussed above chapter, the hash value is very difficult to recover and it helps to ensure the integrity of the source URL. Also, the hash value is in an unreadable form, there is no problem of information being theft even it is exposed. It also ensures the privacy by hiding the original information.

As discussed in the key pair Section 5.1.1, the consequences of a static private key can be solved by salt. Salt is an additional input to the hash algorithm which is hashed along with the source URL. Here, the implementation uses a timestamp as salt in the encryption process. The current timestamp is taken before the source URL is hashed and it is concatenated with URL to produce the hash value. The current timestamp as input is different for each time of hashing or for each particular user, though the source URL is same. The hash values thus created will be identical for each hashing or hashing for

the particular user. It means an identical hash value is encrypted to produce an identical signature. Hence, the problem of the same signature is solved. The salt is also sent along with the signature to the other parties where verification is required.

The digital signature is created with encrypting the hash value of source URL and a timestamp using the static private key in an origin server. The digital signature, salt and source URL is sent to the agent receiving platform. The receiver platform verifies the source URL is of authenticated source using the static public key of the framework. It also checks the source URL authenticity, integrity and non-repudiation. However, the hash algorithm also ensures the integrity of source URL. The authentication and non-repudiation of the source URL are possible with encrypting the hash value to produce a signature. The signature is only verified with the another key pair of the static private key i.e. static public key. The verification on the receiver side ensures that source URL come from the trusted origin server. Similarly, the origin server cannot deny that its source URL and other entities cannot claim for that source URL.

5.1.3 RSA library

The implementation uses the RSA library that helps in encryption and decryption. The RSA library used in implementation is of Herbert Hanewinkel [41]. The library does RSA encryption using *function RSAencrypt()* and RSA decryption using function *function RSAdecrypt()* from *rsa.js* file. The RSA library is of two type, one for server and another for the client in the mobile agent framework. The server side library has *bas64.js, hex.js* and *rsa.js* while client side has *base64.js, hex.js* and *sha256.js* [41].

5.2 Proof-of-concept implementation

From above discussion, this implementation focuses on securing source URL i.e. *auri* of an origin server. The source URL is encrypted using the static private key in the origin server. The static private key is shown in Listing 5.1.

```
var private_key = "AgCc6r2Tq3ouG3lCHkKYAqpmQj3nTHdwcM2JtbbHzj4FaV
       ACGBrvovoNRJGr/ffhRboCq8aqMJCE9nRb+N8r1op7Af9uw
       9Ec8YNr1yhq6C8B48OTeg2UNfn1BFTZrXH2c3cS39I52vPB4
       heEaS4A0NmsKMedU2qsVd/QbmXM2I19yKDx";
```

Listing 5.1: An example of static private key

The implementation works with the life cycle i.e., an agent stages from the origin to the exit of a mobile agent [2]. When an origin server starts, it also loads the RSA server library. The browser (In Figure 5.1, Browser 1) requests the agent using HTTP request with the origin server. The RSA client library is also downloaded that has been added to the *configuration.js*. When an agent is downloaded from the origin server, an instance of

an agent is created using the source URL. Also, the source URL of that agent is encrypted to create a digital signature. The encryption process uses *function encrypt()* (Listing 5.2). In the *function encrypt()*, source URL is concatenated with the time stamp i.e. salt. The concatenation result is the input to the hash algorithm i.e. "SHA-256" [43] to produce hash value (Figure 5.2, step 2). The hash value is encrypted using the static private key that produces the digital signature (Figure 5.2, step 3). The digital signature is bound in JavaScript Object Notation (JSON) format that includes source URL and the time stamp (Figure 5.2, step 4). Each of the agent instances has its own digital signature that moves with the agent. The code for *function encrypt()* is given below:

```
  function encrypt(source_url) {
// obtaining current time to use it as salt in hashing
  var currenttime = Math.round(Date.now() / 1000) ;

// hashing origin url and currenttime as salt
  var hashing = SHA256(source_url+currenttime).toString();

// extracting first 40 string from the hashvalue
  var hashtext = hashing.substr(0,39);

// encryption process
  var enc=rsa1.RSAencrypt(hashtext,private_key);
  var encryptedhex=hex.s2hex(hex.b2s(enc));

// sending values such as digital signature,original url of server ,
   currenttime from server to client as json
  var jsonx = '{"digitalsig":"'+encryptedhex+'","originalmsg":"'+
     source_url+'","currenttime":"'+currenttime+'"}';
   return jsonx;
}
```

Listing 5.2: An example of function to encrypt the source address.

An agent moves from server to browser and vice versa. On the successor platform, the agent has to verify its source URL is of the authentic origin server before the agent is fully downloaded. The agent is verified when it moves from server to client and client to server.

- **When an agent moves from server to client**

  The agent source URL is verified using the digital signature and static public key of the framework. The digital signature is decrypted using the public key that gives the hash value , let it be HASH A (Figure 5.2, step 5). On the other hand, source URL is concatenated with the time stamp which is the input to the hash algorithm, that produces another hash value, let another hash be HASH B. Both hashes HASH A and HASH B are compared. If the hashes are identical to eachother, it verifies

that the source URL is of legitimate origin. However, if hashes are non identical to eachother, the process halts giving the *invalid origin* message (Figure 5.2, step 7). The code for the authentication of source URL when agent moves from server to client is given in Listing 5.4:

```
  var public_key="AgCc6r2Tq3ouG3lCHkKYAqpmQj3nTHdwcM2JtbbHzj4FaV
          ACGBrvovoNRJGr/ffhRboCq8aqMJCE9nRb
          +N8r1op7AAUR";
```

Listing 5.3: An example of static public key

```
// parsing variable jsondata to convert into javascript value
  var myvar = JSON.parse(jsondata);

// assigning digital signature, source url and currenttime.
  var sig = myvar.digitalsig;
  var originalmessage = myvar.originalmsg;
  var currenttime = myvar.currenttime;

// decryption process
  var enc=s2b(hex2s(sig));
  var dec=b2s(RSAdecrypt(enc, public_key));
  var decrypted=dec.substr(0, dec.length-1);

//hashing original message and currenttime as salt in client.
  var origin_add = src.substr(0,21)
  var hashing = CryptoJS.SHA256(origin_add+currenttime).toString()
      ;

// extracting first 40 string from the hashvalue
  var hashvalue = hashing.substr(0,39);

// comparing decrypted value and evaluated hash value
  if (decrypted == hashvalue) {
    console.log("valid orgin");

     } else {
        document.write("invalid_origin");

     }
```

Listing 5.4: An example of authentication process in client.

- **When an agent moves from client to server**

  The agent is pushed from the browser to agent server with HTTP POST (Figure 5.2, Browser1 to Agent server 1). As discussed in Chapter 3 an agent is uploaded in the frozen state with serialization(Figure 5.2 step 8). The serialization is in JSON

format that contains URIs of agent functionality i.e. addresses (*auri, huri*), *id* and other relevant parts [1; 2]. The security implementation adds a JSON that contains a digital signature, agent source URL and timestamp in serialization. This additional JSON is previously carried by the agent itself when it moves to Browser 1 from an Origin server. The serialization agent description looks like (Listing 5.5):

```
    {"auri": "http://localhost:8890/ClockExample.js",
    "huri": "http://localhost:8890/ClockExample.html",
    "id": "577524",
    "curi": "",
    "digitalsign":{"digitalsig":"f1c6ef3babeabc051a4ca8d7c10318
            19aaf739ce502cb5bdbe5534586fc9936692bfcbdb4
            c37dc4171f31e391424fb9b175e744fb3bdf250c1dd
            faf851e1859b",
            "originalmsg":"http://localhost:8890",
            "currenttime":"1446045577"},
                    "variables": {},
                    "memory": {"val": 36}
      }
```

Listing 5.5: An example of serialized agent having digital signature [2]

The Agent server 1 receives serialization and the frozen agent. Usually, the Agent server 1 fetches the JavaScript code using the serialization variable *auri* to restart the agent[14; 35]. Before that, the source URL of serialization should verify its authenticity. The process of authenticity is similar above process as decrypting the signature with the static public key and evaluating the hash from the source URL. Similarly, the hashes are also compared and only the identical hashes proves that source URL is legitimate. Also, the programs halts giving *invalid source*, if hashes are not identical to eachother. This is shown in Figure 5.2, step 9, 10, and 11. The authentication of source URL when the agent moves from client to server is given below:

```
// assigning digital signature, origin url and currenttime.
    var sig = Servertoclient.digitalsig;
    var originalmessage = Servertoclient.originalmsg;
    var nowtime = Servertoclient.currenttime;

// decryption process
    var enc=hex.s2b(hex.hex2s(sig));
    var dec=hex.b2s(rsa1.RSAdecrypt(enc, public_key));
    var decrypted=dec.substr(0, dec.length-1);

// hashing original message and currenttime.
    var origin_add = src.substr(0,21);
    var hashing = SHA256(origin_add+nowtime).toString();
```

```
// extracting first 40 string from the hashvalue
    var hashvalue = hashing.substr(0,39);

// comparing decrypted value and evaluated hash value
   if (decrypted == hashvalue) {
         console.log("valid orgin");

      } else {
        console.log("invalid origin");
        process.exit();
     }
```

Listing 5.6: An example of authentication process in server.

The changed architecture of life-cycle of HTML5-based mobile agent is given below:

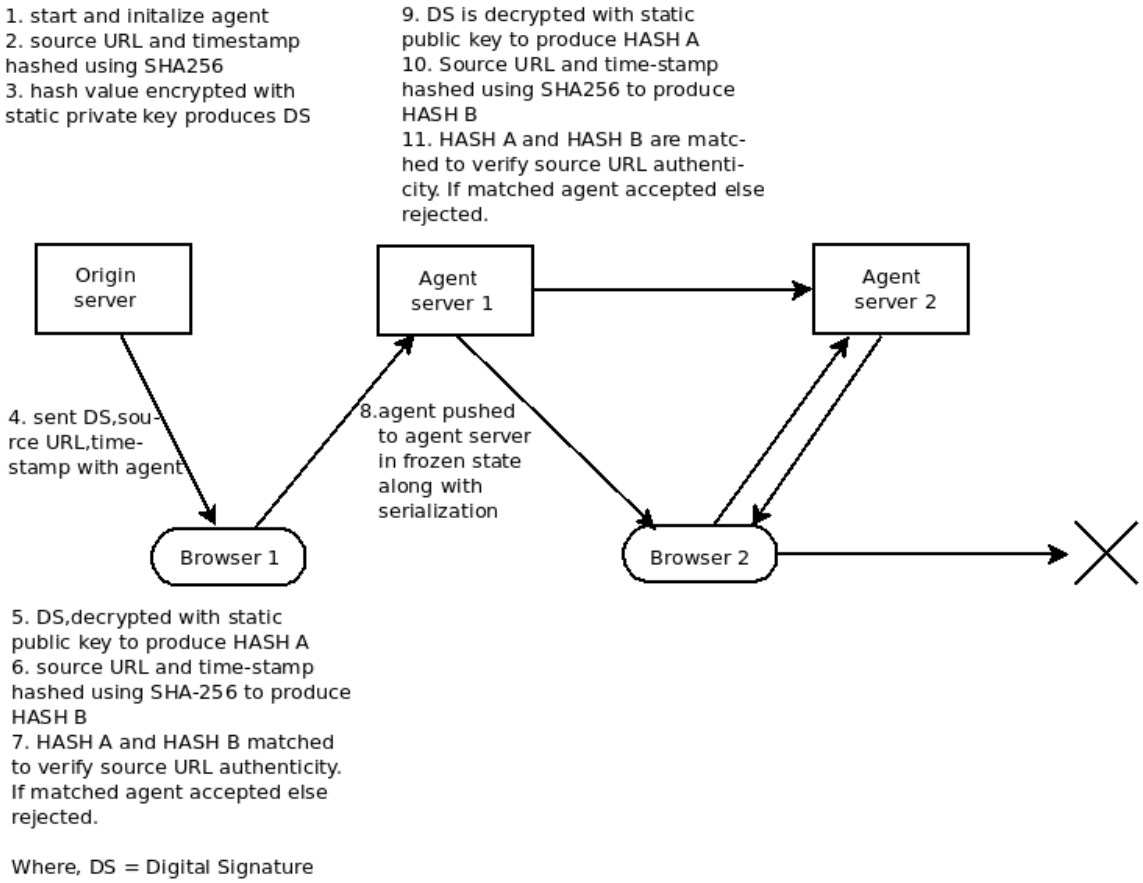

Figure 5.1: Implemenation of security model in HTML5-based mobile agent framework [1].

5.3 Security solution for threat prevention

As discussed earlier in Chapter 4, security threats are categorized into four types on the basis of mobile agent and agent platform. The security implementation discussed in Section 5.2 gives the solution for two threats. They are the agent attacking the platform and the platform attacking the agent.

The primary part of the solution discussed above is centered on preventing the unnecessary modification of the source URL i.e., *auri* of an agent. The implementation uses signing technique to prevent the modification of source URL. The signing technique is used because the agent framework is developed in JavaScript language that lacks the sandboxing. Sandboxes are used when the executables code comes from unknown or untrusted sources and allow the user to run untrusted code. It creates an environment in which there are strict limitations on what resources the agent can request or access. Also, the self-modifiable characteristic of JavaScript language i.e., JavaScript can modify its code as it runs, can be the problem. With the self-modifiable characteristic agent code and platform, the code may change itself when they run. However, the self-modifying characteristic is not always bad but sometimes it may change the underlying run-time or other programs in the framework [52; 53; 54]. The signing technique uses the digital signature that employs the hash algorithm to confirm the authenticity of source URL. It also verifies that source URL has not been modified when the agent moves in different platforms (Section 4.3, Figure 4.2). Also, it ensures that the origin server is non-malicious. Furthermore, the agent source URL is verified in every platform it passes, so it can be claimed as trusted source URL. However, the implementation does not maintain any trusted list. So, a trusted source URL signifies a trustful origin and non-malicious agent.The solution for the treats on agent and platform is described below:

5.3.1 Solution for agent attacking platform/host

The solution for the agent attacking the platform/host refers to the security of the execution environment i.e., platform. As discussed above the source URL is the entry point of an agent in the platform. The source URL is verified before an agent goes to the running state i.e. executable state, in any platform. The method of verifying the source URL will control and prevent untrusted agent trying to enter into the framework. The untrusted agents may be malicious and harmful for the framework. The verifying technique makes the platform free from the malicious agents that try to consume enormous resources and denies to share. The restriction on the malicious agents serves as an access control mechanism to unauthorized access and interference to the platform information. Sometimes the malicious agents may behave as the trusted agent and destroy the reputation earned by the trusted agent. The security solution controls the problem of masquerading and eavesdropping of a platform.

A verified source URL signifies that the origin server it points produces the trusted agent that will not be harmful to the platform and other agents in the platform. The platforms have to communicate with each other to get and transfer the agents. So a proper verification will ensure that a particular communicating platform will send trusted agents. This solution also serves as non-repudiation, as origin server will not be able to deny its agents. Furthermore, an agent can have a secure movement with the help of verification in each platform.

5.3.2 Solution for platform/host attacking agent

The solution for the platform/host attacking the agent refers securing the agent and its important information from the malicious platform. An agent moves to the different platforms, where platform may be descent or malicious. As discussed above source URL is the most important information of an agent, the malicious platform may tamper or reuse the source URL for the malicious activities.

The current security implementation will encrypt the source URL to make the signature. The signature is exposed but the information in it is hashed and then encrypted with a static private key that is hidden in the agent origin server. A small modification in the signature will not verify itself in the an agent receiving platform. Sometimes the malicious platform manages to create a new signature with its own source URL that points malicious platform. The new signature is used instead of the legal signature. However, the verification process in the current security implementation helps to block such agent to enter into the platform. The verification uses a static public key that helps to find out the corrupted, modified or changed signature.

Thus current security implementation helps to get rid the problems like eavesdropping and spying on the agents. Finding security channels and interception of an agent after knowing their weakness will be decreased. It will minimize the effect on agent behavior and accuracy of the computation. It also helps to minimize the problems of the masquerading of an agent platform. Furthermore, an agent can get an identification with its unchanged source URL. So any origin platform will not deny taking responsibility of a particular agent carries an address of that platform i.e. non-repudiation.

6. EVALUATION AND FUTURE IMPLEMENTATION

This chapter describes the evaluation of the current security implementation of Chapter 5. It compares the current implementation with the security measures described in Chapter 4. It also discusses how current implementation is the security solution of the HTML5-based mobile agent framework. Finally, it describes the shortcomings and future implementation in the current solution.

6.1 Possible security measures vs Security implementation

As discussed in above chapters, security is a primary issue in mobile agent technology. There are various security measures available for maintaining security, some are discussed in Chapter 4. The security measures depend upon the threats in mobile agent technology i.e., the agent attacking the platform and the platform attacking the agent. Similarly, the security implementation presented in Chapter 5 also depends upon the threats, security objectives and architecture of a mobile agent paradigm. The objective of the implementation is developing and promoting the secure mobile agent. This implementation is inspired from the security measures proposed in Chapter 4. It uses some techniques of Code signing and environmental key generation. Similar, to the concept of code signing, the security implementation uses a digital signature employing public key cryptography and hash algorithm. The digital signature is responsible for confirming authenticity, originality and integrity of remote information. It verifies that the information has not been corrupted after signing. This security implementation also uses a technique of environmental key generation. The hash generated from the decryption of the digital signature and another hash is evaluated by using SHA-256 algorithm over source URL are compared. Both hashes should be identical to each other as an environmental condition that is to be met in the environmental key generation.

The security implementation uses the salting technique which makes a unique security solution than the security measures proposed in Chapter 4. The implementation uses a timestamp as the additional input to the source URL that serves as salt. The time stamp is concatenated with source URL to yield unique hash value. In the implementation, the unique hash value is encrypted with the static private key to create the digital signature. It means that the digital signature of agents created from the same origin will not be identical to eachother. Sometimes if any malicious entity manages to use a signature, it will not affect on the other agents from the same source. Another unique feature of

the implementation is the trust created by using the static key pairs i.e. static private and static public key. The private key is kept secret in the origin and it is used only for the encryption. Similarly, the public key remains in the framework used only for decryption. The interesting part is the key pairs have some mathematical relationship and the encrypted messages are only decrypted with the public key of that pair. So the agent receiving platform can trust over the source address of the particular agent that agent does not affect the platform and other agents.

The android application store also uses the public key cryptography to maintain security, however the use of keys is somehow different from current security implementation. The android applications are stored in a marketplace such as a google play, from where applications can be downloaded in the client device. In android, uploading an application in a marketplace is called signing an application while a verifying application is called license verification. The application uploading in marketplace takes place in two modes, debug mode and release mode. The application is in debug mode automatically when a project is run or debug. This mode is for developers so that developers can run, debug and make changes in their applications. Similarly, when an application is ready to be downloaded in client device then it is in release mode. A key(it is called private key in android) is generated from a certificate that contains information about the application such as app developers, organization etc and the key is stored securely in a keystore. The key is used to upload an application in the marketplace and further that application is updated using the same key [55]. For the verification of the applications that are stored, the google play service generates RSA private and public key. This key pair is associated with the applications but it are not the same key that is used for uploading the applications. The public key is exposed and embed with the application and private key is kept secret. when an application request for the verification i.e. license check, the google play signs response with the private key of that application, which is verified with the public key that is provided in the application [56].

The security measures for the open source web operating system (OS) like Firefox and Tizen uses the signing technique. Firefox and Tizen have packaged web application. The packaged web application mean all the necessary files that are required for the installation is packaged into the zip package. When an application is installed the zip package is downloaded on the device. For Firefox and Tizen, the package contains a manifest and configuration file which list the APIs that application intends to use. A Firefox manifest is the root directory that contains JSON file that has information such as the application name, description, the developer and the icon. Similarly, a Tizen configuration file contains the application name, description, licenses, developer information and other list of required features. In both OSes, the packages are signed before they are transferred to the device. When the current security implementation is compared with them, the source address of the agent is signed while the package is signed in open source OSes. The source

address in the current implementation is signed in the agent origin server while, in a case of Firefox, the package is signed in the marketplace/ distributor. While in a case of Tizen package is signed by the developer as well as store. Both OSes maintains a trusted source list in manifest and configuration file but the current implementation does not maintain any trusted source list. Similarly, an application is installed when the signature is valid in the device otherwise in case of invalid signature application is not installed. The application verification in the client is similar to the android, where marketplace generates a key pair for each of the applications stored in it. The public key is exposed and attached with the application and private key is kept secret. When an application is requested for the verification, the response is signed with the private key of that application. Finally, the signed response is verified with the public key of the application [46; 47].

Though the open source OSes, Firefox and Tizen uses the signing technique for maintaining the security, it has other features as well. Both OS use content security policy (CSP), that protect from the attack like cross scripting and data injection. The cross scripting refers to the client side code injection attack where the attacker executes malicious script into the legitimate websites [48]. Similarly, there are level of security, for example Tizen has public, the partner and the platform. The level decides the type of signature to be distributed to the users. The signature is installed with the trusted signature and privilege is given according to the level [46; 47].

6.2 Proof-of-concept as security system in mobile agent

The current architecture of mobile agent requires a proper security system as security is a primary issue of the mobile agent development. The proof-of-concept presented in Chapter 5 overcomes the security related issue of the mobile agent. The objective of this implementation is the development and promotion of the secure agent in a mobile agent framework. The implementation ensures that the source address of an agent has not been modified when it moves. The agent moves through the various platform and also prove itself as a non-malicious with the current implementation. Similarly, the platform can be assured that the residing agents came from the legitimate sources. This assurance among the mobile agent entities is maintained with a digital signature, that moves with the agent. The digital signature uses static key pairs, where the static private key remains secretly in origin server and it is used to generate the digital signature when an agent is created. One can be easily assured that the digital signature can only be verified with the static public key of that particular pair. The method of authentication is defined in the current implementation always helps the legal agents to prove themselves that they come from the legitimate origin and they will not harm on the residing platform.

The source URL of an agent is an identification of the origin, where that particular agent is created. Sometimes, malicious entities are able to change an agent identity, the authentication method will help to find out such agents. So, the when platform commu-

nicates to get and transfer an agent, it will help to maintain trust between origin platform and currently agent accepting platform. The proper authentication will help the legitimate agent to enter the platform. So, when agent accepted in any platform, it can communicates with other agent of a platform without any obstruction.

Furthermore, the use of salt in the digital signature will make it more secure. The salt helps to create random and the different digital signature for each of the agents. So malicious entities cannot harm other agents by capturing an agent. It also helps to defend from the "dictionary attack" in the network [45]. The use of the hash algorithm as SHA-256, over the source URL and salt, will produce the hash value. The hash value is very difficult to recover, that adds an extra feature to make the overall implementation more secure [44]. Similarly, the implementation uses RSA algorithm encryption and decryption. The RSA algorithm is one of the widely used cryptographic algorithms. It has been used in many protocols like SSH, OpenPGP, S/MIME and SSL/TLS and in software, for example, browsers. It provides the methods of assuring confidentiality, integrity, authenticity and non-repudiation [39].

6.3 Shortcomings and future implementation

Even though implementation has benefits, it has shortcomings that need to be overcome in future. Some are as follows:

- **Public key distribution:** The public key distribution lacks in the security implementation described in Chapter 5. The security implementation uses a static key pair i.e.,static private and static public key for encryption and decryption. The private key remains secretly in the origin server of an agent helps on encrypting the agent important information. The origin file containing should not be allowed to download. On the other hand, the encrypted information of an agent is only can be decrypted with a public key of a particular pair. So, without the proper public key distribution authentication of the encrypted information is not possible. An agent cannot be verified that it comes from the trusted origin server. In this version of security implementation, the public key remains in the framework. The public key should be distributed from the origin server. It is necessary to be properly distributed in the mobile agent framework so, that malicious entities cannot attack legitimate members of the mobile agent framework.

- ***huri* is not included to create digital signature:** The important agent information is signed to create the digital signature when the agent moves in the various platform. In this security implementation version, only the source URL i.e. address where an agent is created is used for the digital signature. The source URL is also denoted by *auri* variable. The platform fetches the JavaScript file from the address pointed by *auri*. Similarly, *huri* is also important agent information as *auri* that has

HTML-based UI. It is important in the browser when a browser requests an agent with the agent server, the content of HTML-file sends to the browser. The malicious entities may change the information of auri to affect the mobile agent. The current architecture needs some modification so, that both *auri* and *huri* information can be signed digitally and preserve the agent from being malicious.

- **Agent cannot find out malicious platform:** Another shortcoming of this version is an agent is not able to find out the malicious platform without residing in that platform. If a platform is malicious, it will attack the agent and tries to modify the information on it. From the agent view, a particular platform is non-malicious, only if an agent from the previous platform is accepted in the successor platform. Since, a malicious platform will attack the source URL and produce malicious agents. These malicious agents cannot verify themselves in the successor platform. Thus, in successor platform malicious agents are rejected and the predecessor platform known as malicious. The possible way out from this problem is to maintain a trusted list of the platform though the agent is restricted in the network.

- **Proper RSA library:** The RSA library used in the security implementation is restrictive. It only supports a few bits of information to be encrypted.It does not support large agent information to be encrypted. So it is better to find or prepare the RSA library according to the architecture of the framework.

7. CONCLUSION

This thesis studies security challenges of the HTML5-based mobile agent framework and includes design and proof-of-concept implementation of a security framework to protect against some common framework. As the mobile agent technology offers a paradigm that has advantages like reduction of network traffic, connection time, bandwidth consumption over the existing traditional approach such as client-server. However, these advantages would be difficult to achieve without the proper security implementation. So, this thesis develops a secure environment for the existing framework developed in TUT.

The security study in this thesis categorizes the threats created by the malicious agent and malicious platform for the framework.The proposed security solution is based on a fact that the executable code of the agent is determined by a source URL pointing to the code. The code should only be originated from a trusted source, and the origin is verified by cryptographically signed origin URL. The solution ensures the source address to be authenticated, non-repudiation, confidentiality, anonymity and integrity. The verification works as the blockade for the malicious agent and the malicious platform that can trick on the agent and platform. Furthermore, the legitimate agent and platform can be ensured to the source address and trust over the entities of that origin will not harm them. The signature is attached to the agent and moves with it, that gives the identification of agent origination. Also, it ensures the agent mobility to be secured, as in every new environment signature is verified.

The major drawback of this thesis is the public key distribution, as the signature verifying platform should get the genuine key. Also, the architecture of HTML5-based mobile agent lacks to includes the source address of the UI of the mobile agent.

REFERENCES

[1] L. Järvenpää, *Development and evaluation of HTML5 agent framework*, Master of Science Thesis, Tampere University of Technology, 2013.

[2] K.Systä, J.P. Voutilainen, A.L. Mattila, T. Mikkonen, *HTM5-based mobile agents for Web-of-Things*, Tampere University of Technology, Tampere, Finland, 2013.

[3] D. Kotz, R. S. Gray, *Mobile agents and the future of the Internet*, Department of comuter Science/ Thayer School of Engineering Dartmouth college, Hanover New Hampshire 03755, 1999, [Online], Availabile at: `http://www.cs.dartmouth.edu/~dfk/papers/kotz:future2/`

[4] C. G. Olivella, *Contributions To Mobile Agent Protection From Malicious Hosts*, Thesis of Doctor of Philosophy in Computer Science, Universitat Autonoma de Barcelona, 2008.

[5] N. Borselius, *Multi-agents system security for mobile communication*, Thesis of Doctor of Philosophy, Department of Mathematics, University of London, 2003.

[6] Lian, *Overview of Mobile Agent Software*, Tryllian.com, 2013, [Online], Availabile at : `http://www.tryllian.com/overview-of-mobile-agent-software/`

[7] J. cao, S. K. Das, *Mobile agents and applications in networking and distributed computing*, Mobile Agents in Networking and Distributed Computing, Chapter 1, 2012.

[8] J.M. Bradshaw, *An introduction to software agents*, In Software Agents, edited by J.M. Bradshaw, PP 3-46, Cambridge, MA: AAAI Press/The MIT Press, 1997.

[9] M. Wooldridge, *Introduction* An Introduction to MultiAgent Systems, PP 1-12, Chapter 1, Copyright © 2009 John Wiley & Sons Ltd, 2009.

[10] D. Sharma, W. Ma, D. Tran, M. Anderson *A Novel Approach to Programming: Agent-based Software-Engineering* School of Information Sciences and Engineering University of Canberra, B. Gabrys, R.J. Howlett, and L.C. Jain (Eds.): KES 2006, Part III, LNAI 4253, pp. 1184 – 1191, 2006.

[11] S. Franklin, A. Graesser, *Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents*, Institute for Intelligent Systems, University of Memphis, 1996, [Online]. Available at: `http://www.inf.ufrgs.br/~alvares/CMP124SMA/IsItAnAgentOrJustAProgram.pdf`

[12] D. B. Lange, *Mobile agent with Java: The Aglet API*, General Magic Inc, 420 North Mary Avenue, Sunnyvale, CA 94086 U.S.A, Mitsuru Oshima IBM Tokyo Research Labratory, 1623-14 Shimotsuruma, Yamato-shi Kanagawa-ken 242, Japan, 1998.

[13] M.Alfalayleh and L.Brankovic, *An Overview of Security Issue and Tecnhiques in Mobile Agents*, The School of Electrical Engineering and Computer Science, The University of Newcastle NSW 2308 Austrilia,[Online], Available at : `http://sec.cs.kent.ac.uk/cms2004/Program/CMS2004final/p2a3.pdf`

[14] W.A.Jansen, *Countermeasures for Mobile Agent Security*, National Institute of Standards and Technology, Gaithersburg, MD 20899, USA.

[15] J. Ylitalo, *Secure Platform for Mobile Agents*, Department of Computer Science, Helsinki University of Technology, 2000, [Online], Available at : `http://www.tml.tkk.fi/Opinnot/Tik-110.501/1999/papers/mobileagents/`

[16] G.C. Kessler, *An overview of cryptography*, shorter, edited version of this paper appears in the 1999 Edition of Handbook on Local Area Networks, 2015, [Online], Available at : `http://www.garykessler.net/library/crypto.html`

[17] W. Jansen and T. Karygiannis *Mobile Agent Security*, Computer Security, National Institute Of Standards and Technology(NIST)special publication 800-19, U.S., Department of Commerce, 1999, [Online]. Available at: `http://csrc.nist.gov/publications/nistpubs/800-19/sp800-19.pdf`

[18] introduction to cryptography,Copyright © 1990-1999 Network Associates, Inc. and its Affiliated Companies. All Rights Reserved. see Introtocrypto.

[19] G. McGraw and E. Felten, *Securing Java, Beyond the Sandbox : signed code ad Java 2, Section 4 –Trust*, 1999, [Online], Available at : `http://www.securingjava.com/chapter-three/chapter-three-4.html`

[20] W. M. Farmer, J. D. Guttman and V. Swarup,*Security for Mobile agents: Authentication and State Appraisal* The MITRE Corporation, 202 Burlington Road, Bedford, [Online], Available at : `http://web.cs.wpi.edu/~guttman/pubs/sema-esorics96.pdf`

[21] M. Vella, *Introduction to cryptography* Foundations of symmetric and asymmetric key cryptography,Computer Science Dept., Faculty of ICT, University of Malta, 2013.

[22] A. Lingnau, O. Drobnik, P. Dömel, *An HTTP-based Infrastructure for Mobile Agents*, Johann Wolfgang Goethe-Universität Frankfurt, Germany, [Online], Available at:http://www.w3.org/Conferences/WWW4/Papers/150/

[23] Creation common Attribution 2.5 License, *Introduction to Distributed System*, Google, 2007, [Online], Availabile at : https://courses.cs.washington.edu/courses/cse490h/07wi/readings/IntroductionToDistributedSystems.pdf

[24] Y. Aburabie, Dr. L. Tawalbeh, *Intrusion Detection and mobile code presentation*, New York Institute of Technology(NYIT),Presentation slide,2006.

[25] www.redhat.com, *Introduction to public-key cyrptography*, chapter 1,Red Hat, Inc.Copyright © 2015, [Online], Available at: https://access.redhat.com/documentation/en-US/Red_Hat_Certificate_System/8.1/html/Deploy_and_Install_Guide/Introduction_to_Public_Key_Cryptography.html

[26] S. Turunen, *Production of an HTML5 Agent Framework*, Master of Science Thesis, Tampere University of Technology, 2014.

[27] O.M.Paracha, *A survey of countermeasures for protection of mobile agent systems*, A Security Framework for Mobile Agents systems, Chapter 4, 2006.

[28] T. Ishida(Ed.), Department of social informatics, Kyoto University, Japan, *Multi-agents platforms*, First Pacific Rim International Workshop on Multi-Agents PRIMA'98, Singapore, 1998.

[29] D. B. Lange, M. Oshima, *Seven good reason for Mobile Agents*, Communication of the ACM, 1999, PP 88-89.

[30] B. Kalisk, *The Mathematics of the RSA Public-Key Cryptosystem*, RSA Laboratories

[31] S. Robles, *Mobile Agent Systems and Trust, a Combined View Toward Secure Sea-Of-Data Applications*, Thesis of Doctor of Philosophy in Computer Science, Universitat Autonoma de Barcelona, PP 9-10, 2002 .

[32] I. Reinhartz-Berger, D. Dori, S. Katz, *Modeling code mobility Paradigms in OPM/Web*, Technion ,Israel Institute of Technology, [Online], Available at : https://www.research.ibm.com/haifa/info/ple/present/ibmPresentation.pdf

[33] A. Carzaniga, G. Pietro Picco, G. Vigna, *Designing Distributed Application with Mobile Code Paradigms*, Politecnico di Milano, [Online], Available at : http://www.inf.usi.ch/carzaniga/papers/icse97.pdf

[34] H. S. Nwana, Intelligent Systems Research,Advanced Applications and Technology Department BT Laboratories, Martlesham Heath, U.K. *Software Agents: An Overview*, Knowledge Engineering Review, Vol. 11, No 3, pp. 205-244, October/November 1996.Cambridge University Press, 1996.

[35] N. Moraitakis, *Intelligent Software Agents, Application and Classification*, Imperial College of Science an Technology and Medicine, Department of Computing and Electronic Engineering, 1997, Available at :`http://www.doc.ic.ac.uk/~nd/surprise_97/journal/vol1/nm1/`

[36] M. Obitko, *FIPA Compliant Agents*, Department of Cybernetics, Faculty of Electrical Engineering Czech Technical University, Prague, Czech Republic, 2007, Available at : `http://www.obitko.com/tutorials/ontologies-semantic-web/fipa-compliant-agents.html`

[37] P. Braun, W. Rossak *Mobile agent migration*, Mobile agents, Basic concepts, Mobility models and the Tracy Toolkit, Morgan Kaufmann publisher and dpunkt.verlag Chapter 3, 2005.

[38] K. Systä, T. Mikkonen and L. Jarvenpaa, *HTML5 Agents: Mobile Agents for the Web*, Department of Pervasive Computing, Tampere University of Technology, Tampere Finland, 2014.

[39] M. Rouse, TechTarget *RSA algorithm (Rivest-Shamir-Adleman) definition* copyright TechTarget 2000-2015, [Online], Available at: `http://searchsecurity.techtarget.com/definition/RSA`

[40] R.L. Rivest, A. Shamir, and L. Adleman *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems* Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, 1977

[41] H. Hanewinkel, 2005,[Online], Available at:`http://www.hanewin.net/encrypt/rsa/rsa.htm`

[42] B. Steyn, How RSA Works With Examples,Copyright 2015, Barry Steyn Canada, 2012, [Online], Available at : http://doctrina.org/ How-RSA-Works-With-Examples.html

[43] www.wikipedia.org, *SHA-2*, Wikimedia Foundation, Inc.2015, [Online], Available at :`https://en.wikipedia.org/wiki/SHA-2`

[44] www.wikipedia.org, *Salt (cryptography)*, Wikimedia Foundation, Inc.2015, [Online], Available at :`https://en.wikipedia.org/wiki/Salt_(cryptography)`

[45] www.wikipedia.org, *Dictionary attack*, Wikimedia Foundation, Inc.2015, [Online], Available at : https://en.wikipedia.org/wiki/Dictionary_attack

[46] Mozilla Developer Network and individual contributors, *Firefox OS security overview*, Copyright © 2005-2015, [Online], Available at: https://developer.mozilla.org/en-US/docs/Mozilla/Firefox_OS/Security/Security_model

[47] O.Gadyatskaya, F. Massacci, Y. Zhauniarovich *Emerging Mobile Platforms: Firefox OS and Tizen*, Department of Information Engineering and Computer Science, University of Trento, Article in computer 2014.

[48] www.acunetix.com, *Cross-site Scripting (XSS) Attack*, Copyright © Acunetix, 2015, [Online], Available at: https://www.acunetix.com/websitesecurity/cross-site-scripting/

[49] R. Oppliger *Asymmetric encryption systems*, Contemporary Cryptography, Second edition, Chapter 10, pp. 284, © Artech house, 2011.

[50] www.mathsisfun.com, *Relatively prime*, Copyright © 2014 MathsIsFun.com [Online], Available at: http://www.mathsisfun.com/definitions/relatively-prime.html

[51] www.arduino.cc, *Modulo*, Copyright © 2016 Arduino, [Online], Available at: https://www.arduino.cc/en/Reference/modulo

[52] C.G. Harrison, D.M. Chess, A. Kershenbaum *Mobile Agents: Are they a good idea?*, Mobile Object Towards the Programmable Internet, Volume 1222, 1997, pp. 25-45.

[53] http://www.webopedia.com, *Sandbox*, Copyright © Copyright 2016 QuinStreet Inc., [Online], Available at: http://www.webopedia.com/TERM/S/sandbox.html

[54] http://www.i-programmer.info, *Javascript Jems - Self Modifying Code*, Copyright © 2016 i-programmer.info., [Online], Available at: http://www.i-programmer.info/programming/javascript/989-javascript-jems-self-modifying-code.html

[55] http://developer.android.com/, *Signing Your Applications*, Creative Commons Attribution 2.5 Generic License, [Online], Available at: http://developer.android.com/tools/publishing/app-signing.html#overview

[56] http://developer.android.com/, *Adding Licensing to Your App*, Creative Commons Attribution 2.5 Generic License, [Online], Available at: `http://developer.android.com/google/play/licensing/adding-licensing.html`