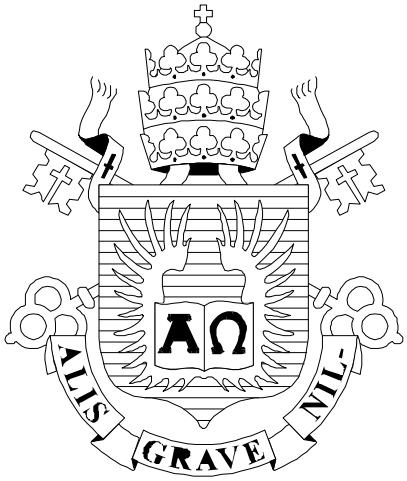

PUC

ISSN 0103-9741

Monografias em Ciência da Computação
n° 02/16

An Architecture to Mitigate Buffer Overflow Attacks, using Multi-agent System Concepts

Marcio Ricardo Rosemberg
Francisco José Plácido da Cunha
Carlos José Pereira de Lucena
Daniel Schwabe
Marcus Poggi

Departamento de Informática

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO
RUA MARQUÊS DE SÃO VICENTE, 225 - CEP 22451-900
RIO DE JANEIRO - BRASIL

Monografias em Ciência da Computação, No. 02/16 ISSN: 0103-9741
Editor: Prof. Carlos José Pereira de Lucena June, 2016

An Architecture to Mitigate Buffer Overflow Attacks, using Multi-agent System Concepts*

Marcio Ricardo Rosemberg [1] Francisco José Plácido da Cunha[1] Carlos José Pereira de Lucena[1] Daniel Schwabe[1] Marcus Poggi[1]
[1]Departamento de Informática – Pontifícia Universidade Católica do Rio de Janeiro (PUC-RIO)

{mrosemberg, fcunha, lucena, dschwabe, poggi} @inf.puc-rio.br

Abstract: Most times, services or daemons are written in C and, therefore, are subjected to Buffer Overflow Attacks and other exploits that the C language is vulnerable. When such services perform authentication, and are successfully exploited, the attacker may gain access to user credentials, session keys and even the Private Key of a digital certificate. In this work, we propose an architecture for modeling and implementation of services and client applications, using Multi-Agent Systems concepts to mitigate the damage an attacker could inflict on a system. Our approach also improves the resilience to Denial of Service Attacks, since an intelligent agent can learn from past experiences and be pro-active in the defense of a system under attack.

Keywords: Agents, Security, MAS, Buffer Overflow, Heartbleed

Resumo: Na maioria das vezes, serviços ou daemons são escritos em C e, portanto, estão sujeitos a ataques de Buffer Overflow e outras vulnerabilidades que a linguagem C é susceptível. Quando tais serviços envolvem autenticação e são explorados com êxito, o atacante pode obter acesso a credenciais de usuário, as chaves de sessão e até mesmo a chave privada de um certificado digital. Neste trabalho, propomos uma arquitetura para modelagem e implementação de serviços e aplicações de cliente, usando conceitos de sistemas multiagentes para atenuar o dano que um invasor poderia causar em um sistema. Nossa abordagem também melhora a capacidade de resiliência a ataques de negação de serviço, visto que um agente inteligente pode aprender com as experiências passadas e ser proativo na defesa de um sistema sob ataque.

Palavras-chave: Agentes, Segurança, SMA, Buffer Overflow, Heartbleed

* This work has been sponsored by the Ministério de Ciência e Tecnologia da Presidência da República Federativa do Brasil and the CNPQ.

In charge of publications:

Rosane Teles Lins Castilho
Assessoria de Biblioteca, Documentação e Informação
PUC-Rio Departamento de Informática
Rua Marquês de São Vicente, 225 - Gávea
22451-900 Rio de Janeiro RJ Brasil
Tel. +55 21 3527-1516 Fax: +55 21 3527-1530
E-mail: bib-di@inf.puc-rio.br
Web site: http://bib-di.inf.puc-rio.br/techreports/

Table of Contents

1 Introduction

Servers run services or daemons to receive requests from clients and return results. However, before disclosing sensitive information, services use authentication protocols, such as the Transport Layer Security (TLS) [1], to authenticate both the client and the server, establishing a secure communications channel during the process[2].

Many Authentication Protocols use asymmetric cryptography to authenticate each endpoint and a negotiated symmetric session key to ensure confidentiality, after the authentication phase. The most important asset of the authentication protocols relying on asymmetric cryptography is the Private Key. If the Private Key is misused or stolen, the authentication is compromised, meaning that any person or entity can impersonate the true owner of the Private Key[3]. Likewise, if session keys are compromised, an attacker could eavesdrop or even steal the session. The perpetrator may cause considerable damage to the system and leave the blame on the attacked user.

Most times, services or daemons are written in C or C++ and, therefore, are subjected to Buffer Overflow Attacks[4] and other exploits that the C language is suitable.

One type of attack that exploited buffer overflows and have caused major damages is the Heartbleed Attack[5]. This attack is capable of stealing user accounts, passwords, session keys and even private keys[6]. One of the reasons Heartbleed achieved such success is because the services that used the affected versions of Open-SSL[I] were designed to run in a single process, storing in main memory the session keys, cached user credentials and the Private Key of the digital certificate.

By distributing a single service into an architecture composed of two agent platforms, we can mitigate the damage a Buffer Overflow Attack or an attack such as Heartbleed could impose on an exploited system. Each agent platform runs under an isolated process. However, only one of the platforms (the insecure platform) is exposed to the Internet or any other network. The other platform (the secure platform), which stores private keys, user credentials and session keys in memory, is not exposed.

A Multi-agent System (MAS) is an excellent paradigm to model an implement security oriented software. MAS is composed of agents. "An agent is a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its design objectives" [7].

According to Wooldridge[7], any system service or daemon could be viewed as an agent. Services run continuously, reacts to the changes in the environment and processes information without user intervention (autonomy).

Having agents doing specific and isolated functions increases security. By definition, agents do not know the entire system [7][22]. Such feature is very well tuned with the concept that information should be disclosed on a need to know basis [8].

Using MAS concepts, we propose an architecture to implement system services or client applications, particularly the ones that need to access sensitive information and, as a result, require authentication and confidentiality. Our approach focuses on two agent platforms running in two different processes. The insecure platform would be

I http://www.openssl.org

exposed to the insecure network via TCP/IP sockets, while the secure platform would run on another process with no direct interfaces.

2 Background

2.1 The Buffer Overflow Attack

Buffer overflows occur when more bytes are written into a memory area (buffer) than the initially allocated size. If an attacker gains control of what is written into this memory area, he can perpetrate buffer overflow attacks [9].

Services or daemons such as http open sockets and receive requests to be processed and returned to the originating client. The goal of the Buffer Overflow Attack is to disrupt the stack pointer (ESP) of the CPU [10]. By injecting code instead of a well formed message, the service will pass the received message as a string to a function to be processed. The memory space in which temporary data is stored is the stack. The stack grows and shrinks dynamically during the process execution. If the received string is greater the allocated size of the buffer, the stack grows in order to hold the amount of data it received. The stack grows from higher memory addresses to lower memory addresses. If such grow reaches the memory area where the returning address to the main program is held, when the functions terminates its processing it will return control to another address. In a successful attack, the process will execute the code injected by the attacker [11].

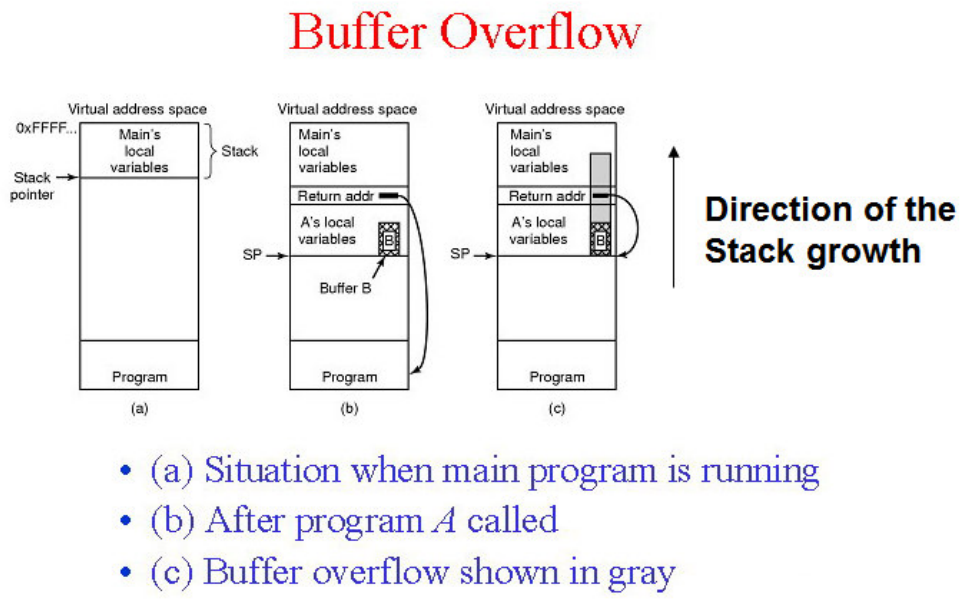

Figure 1 - A Buffer Overflow Attack [II]

Buffer overflow attacks are the main cause for most of the cyber-attacks such as server breaking in and malware insertions [12]. They are the responsible for the vast majority, if not all, the worms [13] and the most commonly reported source of software vulnerabilities [10].

Kundu and Bertino (2011) [9] and Strackx et al. (2009) [12] demonstrated how simple coding can be so vulnerable to Buffer Overflow Attacks.

II Source: https://under-linux.org/content.php?r=5132

Buffers can be allocated on the stack, the heap or in the data segment in C. For arrays that are declared in a function body, the necessary space is reserved in the stack. Buffers that are allocated dynamically are put on the heap, usually via the *malloc* function, while global or static arrays are allocated in the data segment. The array is manipulated by means of a pointer to the first byte. Bytes within the buffer can be addressed by adding the desired index to this base pointer [12].

```
void copy(char* src, char* dst) {
  int i = 0;
  char curr = src[0];
  while(curr) {
    dst[i] = curr;
     i++;
    curr = src[i];
  }
}
```

Figure 2 - A simple C function vulnerable to the Buffer Overflow Attack

Form the code in Figure 2 we may observe that at run-time no information about the array size is available. As a result, the compiler will generate a code that will allow copies beyond the bounds of the array. Therefore, the generated code will write data to the adjacent memory area, stopping only and a \0 character is found indicating a null terminated string [12].

The only way to prevent Buffer Overflow Attacks is by adding security features on C functions or the entire C language such as automatic memory management, strong typing, and overflow checks [15]. However, such security features would also introduce a significant overhead in the programs generated with the listed security features.

In order to mitigate Buffer Overflow Attacks and malware propagation, various products and services are available but none are effective against poor programed software like the Heartbleed bug[5].

2.1.1 Protecting Against Buffer Overflow Attacks

There are two main methods of detecting or preventing buffer overflow attacks, whichever are either over network or on the host machine [11].

- Network Based Intrusion Detection and Prevention Systems (NIDS)
- Host Based Protection Mechanisms

NIDS are based on deep packet inspection. Both the header as well as the payload of the packets are checked with predefined signatures to identify whether it contains malicious data or not [14]. Several commercial products, for instance Symantec Endpoint Protection[III], have incorporated NIDS as part as their anti-virus anti-malware suite. However, as any anti-virus, it can only detect threats present in its knowledge base, even if it uses heuristics to detect unknown threats. New developed attacks may not be detected by the scanning mechanism.

The most widely used Host Based Protection Mechanisms are Data Execution Prevention (DEP) and Address Space Randomization (ASLR) [15]. DEP prohibits memory pages from being both writable and executable, thereby preventing an adversary from injecting and directly executing code. ASLR randomizes code reuse where an attacker

III http://www.symantec.com/endpoint-protection/

will try to use existing functions in the c-library of a system (return-to-libc), which are known to be vulnerable, instead of injecting their own code, and, as a consequence, circumvent DEP. Currently, ASLR implementations randomize the base (start) address of segments such as the stack, heap, libraries, and the executable itself between consecutive runs of the application. The primary objective is to force attackers to guess the location of the functions and instruction sequences needed to successfully launch a code reuse attack. ASLR is known to be vulnerable to brute-force attacks [16] and to memory disclosure attacks [17].

All of the authors referenced in this section agree that Buffer Overflow Attacks are the result of poor constructed software, which employ unsafe string handling functions or inadequate array bounds checking.

2.1.2 The Heartbleed Attack

The Heartbleed Attack is considered the most serious vulnerability revealed. Bruce Schneier (2014) [5] considered the bug "catastrophic" and on the scale of 1 to 10, Heartbleed is 11. More than half a million servers were affected, including Schneier's own web site.

The initial attack is the exact opposite of the Buffer Overflow Attack. Instead of writing more bytes than the buffer could handle, the attacker would send a string of ten bytes, for instance, but will pass 64K as the string length. Because the size of the string and the reported string length were not programmatically checked and matched (the core of the bug), the response was a 64K long string. The original 10 bytes plus all of the remaining bytes of the adjacent memory area.

The attack was designed to exploit the heartbeat extension of TLS. It is a keep-alive feature in which one end of the connection sends a payload of arbitrary data to the other end, which sends back an exact copy of that data to prove that the other end received the original message and everything is fine [18].

The exploit was possible because there were no bounds check prior of a call to the *memcpy* C function, which is vulnerable to Buffer Overflow Attacks.

"An attacker can trick OpenSSL into allocating a 64KB buffer, copy more bytes than is necessary into the buffer, send that buffer back, and thus leak the contents of the victim's memory, 64KB at a time" [19].

The attacker can repeat the exploit numerous times, each time bringing different memory portions of the Server. The server's responses could include user credentials (account names and passwords or passwords hashes), session keys and the Private Key of the server's digital certificate.

Against Heartbleed, none of the Buffer Overflow protections would work, simply because the heartbeat message was a legitimate one, only with the message length parameter altered. Also, the message is sent encrypted with the session key, making it more difficult for an IDS/IPS software to scan for malware.

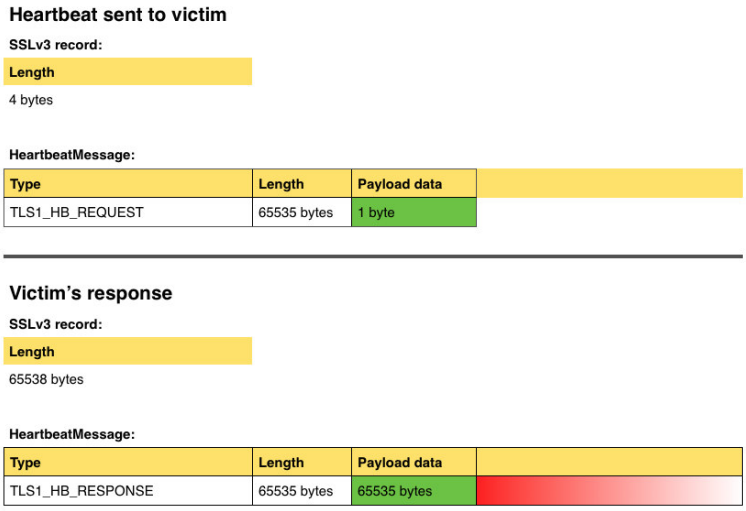

Figure 3 - The Heartbleed Attack [IV]

2.2 Multi-Agent Systems (MAS)

Multiagent systems (MAS) are societies in which autonomous entities (agents), heterogeneous and individually designed, work according to objectives that may be common or different [20]. Agent technologies are well recognized as way of representing and reasoning about complex real world problems in the field of information and communication technologies [21]. An agent is a process or a thread that runs continuously, reacts to changes in the environment and has autonomy to achieve its designed goals. There are two main types of agents: Reactive Agents and Intelligent Agents [22].

2.2.1 Reactive Agents

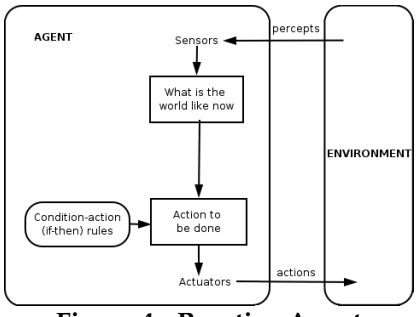

Figure 4 - Reactive Agent

Reactive Agents are computer systems that are capable of autonomous action in some environment in order to meet their design objectives. An agent will typically sense its environment (by physical sensors in the case of agents situated in part of the real world, or by software sensors in the case of software agents), and will have available a repertoire of actions that can be executed to modify the environment, which may appear to respond non-deterministically to the execution of these actions [22].

IV http://regmedia.co.uk/2014/04/09/openssl_haertbleed_diagram.png

2.2.2 Cognitive or Intelligent Agent

An intelligent agent is one that is capable of "***flexible***" autonomous action in order to meet its design objectives[22], where flexibility means three things:

- Reactivity: intelligent agents are able to perceive their environment, and respond in a timely fashion to changes that occur in it in order to satisfy their design objectives
- Pro-activeness: intelligent agents are able to exhibit goal-directed behavior by taking the initiative in order to satisfy their design objectives
- Social ability: intelligent agents are capable of interacting with other agents (and possibly humans) in order to satisfy their design objectives.

One of the widely known architectures for designing and implementing cognitive agents is the belief-desire-intention (BDI) architecture, following a model initially proposed by Bratman [23], which consists of beliefs, desires and intentions as mental attitudes that deliberate human action. Rao & Georgeff [24] adopted this model and transformed it into in a formal theory and an execution model for BDI agents, serving as a basis for the implementation of several BDI agent platforms. However, two limitations of the BDI model are well known [25]: i) Its lack of learning competences; and ii) Its lack of explicit multi-agent systems (MAS) aspects of behavior. The limitations of the BDI model are the subject of what is now known as MAS learning [26] [27], roughly characterized as the intersection of Machine Learning (ML) and MAS.

Although the agents in a MAS can be programmed with behaviors designed in advance, it is often necessary that they learn new behaviors, such that the performance of the agent or of the whole MAS gradually improves [28],[26].

In order to be pro-active, an agent must have a learning module [29]. This module saves its experiences in a Knowledge Base for future reference. For example: an agent may decide to block an IP address if it deduces that such address is the source of an attack. After some time, it may decide to unblock the previously blocked address. However, if the address is believed to be the source of other attacks, the agent may decide to block the address for a longer time and, possibly, to send a message to the IP owner, reporting the attack and ask for verification and correction of the alleged misused address.

Because of the social ability of intelligent agents, they express behaviors according to their goals and their design strategies to achieve their goals. They may express cooperative behavior, meaning they help other agents to achieve their goals or they may express selfish behavior, meaning they do not cooperate with other agents.

In order to express social behavior, each agent should incorporate a learning module to learn and explore the environment. In addition, due to interaction among the agents in a multi-agent system, they should have some sort of communication between them to behave as group [30]. The advantage of multi-agent learning is that the performance of the agent or of the completely multi-agent system gradually improves [31]. For example, Russel and Norvig [43] have conceptually structured generic learning agent architecture as depicted below.

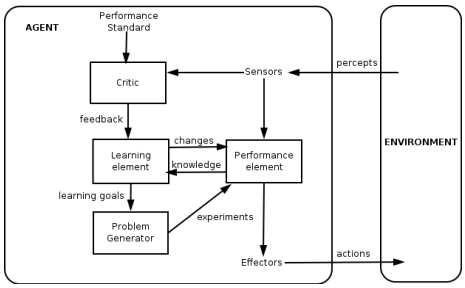

Figure 5 - Cognitive or Intelligent Agent

2.2.3 Trust and Reputation

Trust and reputation concepts are widely used in various fields of computer science, such as evaluation systems, P2P networks, grid computing, game theory, e-commerce, semantic web, software engineering, web services, and recommendation systems [32]. Trust is defined as subjective probability with which agents assess that other agents will perform a particular action, while Reputation is defined as a subjective probability with which agents assess that other agents will provide trustful testimonies [33].

Because agents have autonomy to achieve their design goals, they may even lie (present false information) if it is in their interests to achieve their goals. By doing so, they lose reputation. When an agent's reputation level lowers bellow a certain point, other agents will decide not to trust such an agent anymore.

Because the MAS definition of trust conflicts with the definition of trust in the security domain, one must be very careful to adopt the concepts of Trust and Reputation for agents when designing a system for the security domain.

In our proposal, we decided not to use agent Trust and Reputation concepts.

2.2.4 Developing Multi-Agents Systems

Since agents need to find other agents and establish communications with them, standards have to be specified and followed. FIPA [V] provides the standards for building frameworks to be used for MAS development. FIPA specified an abstract architecture, leaving implementation details and internal architectures to platform developers [34].

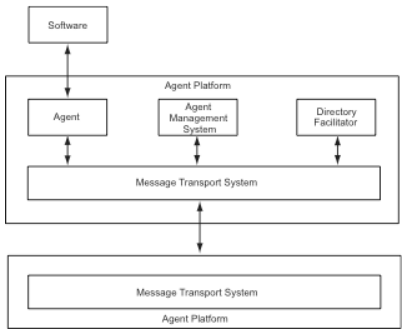

Figure 6 - FIPA reference model of an agent platform

V The Foundation for Intelligent Physical Agents. http://www.fipa.org

The Agent Management System (AMS) is a mandatory module that exerts supervisory control over access to and use of the platform; it is responsible for maintaining a directory of resident agents (white pages) and for handling their life cycle.

The Directory Facilitator (DF) is the optional module that passes on yellow page services to the agent platform.

The Message Transport System (MTS) is another mandatory module, which provides communications between agents. An Agent communicate with other agents by message exchange. The specifications also define the Agent Communication Language (ACL) [34].

2.3 Cryptography Basics

The word Cryptography comes from the Greek words kryptós (hidden) and gráphein (to write) [35]. Cryptography is used to protect sensitive or secret data in a way that unauthorized people or computerized systems are unable to understand or make use of the data. In Computer Science, Cryptography works in 3 steps [36]:

Encryption: the process to cipher the original message. The message could be plain text, an image, a stream of bits, voice data or any form binary data. The Ciphered message C is obtained by applying the function E on the original message M.

$$C = E(M)$$

- Transmission of the ciphered message
- Decryption: the process to decipher the ciphered message back to the original message. The original message M is obtaining by applying the function D on the ciphered message C

$$M = D(C)$$

Figure 7 - Encryption and decryption

Cryptography performs major roles in information security. It helps to enforce Confidentiality, Integrity, Authenticity and Non-Repudiation.

The Mechanics of Cryptography involves a sender (Alice), a receiver (Bob) and sometimes a trusted third party (Trent) [37]. There are cryptographic algorithms that involve more parties.

2.3.1 Algorithms and Keys

Since Encryption and Decryption are functions, they are based on an algorithm. If the security is based on the algorithm, then the algorithm must be kept secret at all costs. If it leaks, everybody that uses the algorithm needs to change it. The solution to this problem is the use of public but strong cryptographic algorithms that use one or more keys to encrypt and decrypt messages. If the keys are compromised, the parties involved just need to change the keys, maintaining the algorithm. When an algorithm uses keys, the encryption and decryption functions are expressed $C=E_K(M)$ and $M=D_K(C)$, respectively, such that $D_K(E_K(M))=M$ holds [38]. Security is based in the size and complexity of the key (the longer and the more complex the better) and the complexity of the algorithm (usually, the more complex the better). Complexity of the algorithm increases the difficulty to write another algorithm capable of decrypting the ciphered message or capable of deducing the encryption key. Complexity of the key increases the difficulty to guess the key in a brute force attack.

2.3.2 Symmetric Algorithms

Algorithms that use the same key to encrypt and decrypt messages are called symmetric algorithms. Alice and Bob must agree on a single encryption and decryption key which would be used by both [39].

The problem with symmetric algorithms is how Alice and Bob negotiate a session key (a symmetric key used in one communications session) in an insecure channel. Unless they agree to meet in person and negotiate the key, there is always the possibility that an eavesdropper listens to the key negotiation and renders the encryption process useless. On the other hand, an active attacker can do much worst. An active attacker can negotiate a session key with Alice and another with Bob. Then, he can decrypt Alice's message, forge another message and send it to Bob. Bob thinks he received an authentic message from Alice and Alice doesn't know Bob received a false message.

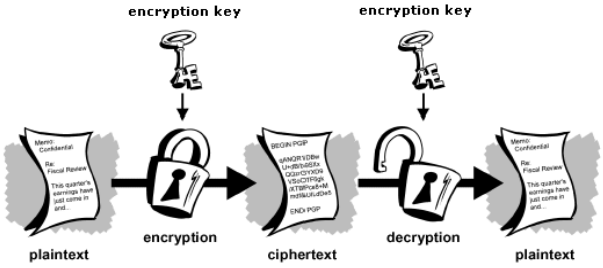

Figure 8 - Symmetric encryption and decryption

2.3.3 Asymmetric Algorithms

Algorithms that use different encryption and decryption keys such that even in possession of one of the keys one cannot calculate the second in a reasonable amount of time are called asymmetric algorithms. One of the keys is the Public Key that can be widely distributed. The other key is the Private Key known only by its owner. Messages encrypted with the Public Key can only be decrypted by the Private Key.

The most widely used asymmetric algorithms are the RSA, El-Gamal and Elliptic Curves [40].

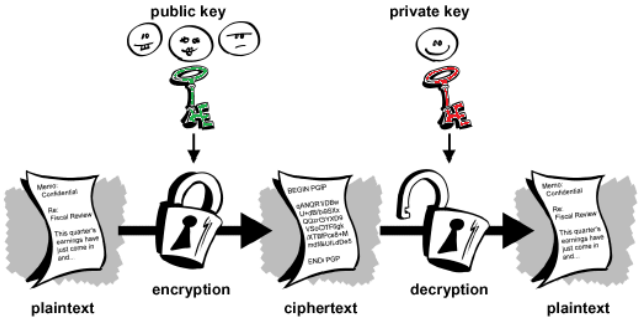

Figure 9 - Asymmetric Algorithms

2.3.4 Digital Signatures

A digital signature is an algorithm designed to validate the authenticity of a digital message. A valid digital signature gives the recipient reason to believe the message the have not been modified while in transit, enforcing Integrity. If the digital signature is bound to an unique person or organization the recipient has reason to believe the message was created by a known sender, such that the sender cannot deny having sent the message, enforcing both non-repudiation and authentication. One way hash algorithms is a good way to provide integrity. If Alice sends a message to Bob with a SHA-128 hash[41] attached to the message and the message is tampered while in transit Bob will calculate the SHA-128 hash of the received message and it will not match the SHA-128 hash supplied by Alice. However, a one-way hash algorithm does not enforce non-repudiation or authentication.

Alice enforces authenticity by generating a hash of the message and encryption of the hash with her Private Key, digitally signing the message. Alice provides authentication and non-repudiation, because the digital signature can only be verified by Bob with Alice's Public Key and a new hash computed of the received message. If the verification process fails, either the message lost integrity during transmission or it was tampered by an attacker. However, if the verification processes succeeds, the message is authentic and non-repudiation may be assured [42].

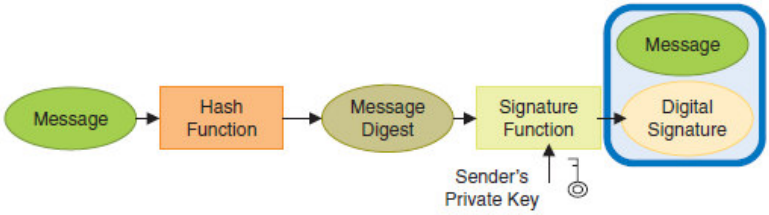

Figure 10 - Digital Signature generation

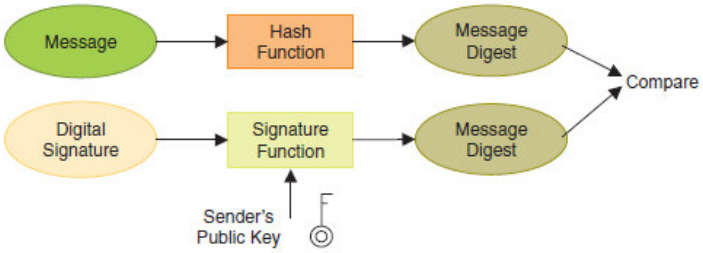

Figure 11 - Digital Signature verification

3 The Proposed Solution

We propose an architecture composed of two FIPA compliant agent platforms. The agent platform exposed to the insecure network is the Insecure Platform. The agent platform not exposed to the insecure network is the Secure Platform.

Each agent platform must run in an independent process. Inter-process Communications (IPC) between the platforms must be done using either named pipes or message queue. Message queue should be the preferred method because it increases modularity, scalability and, most importantly, security [43].

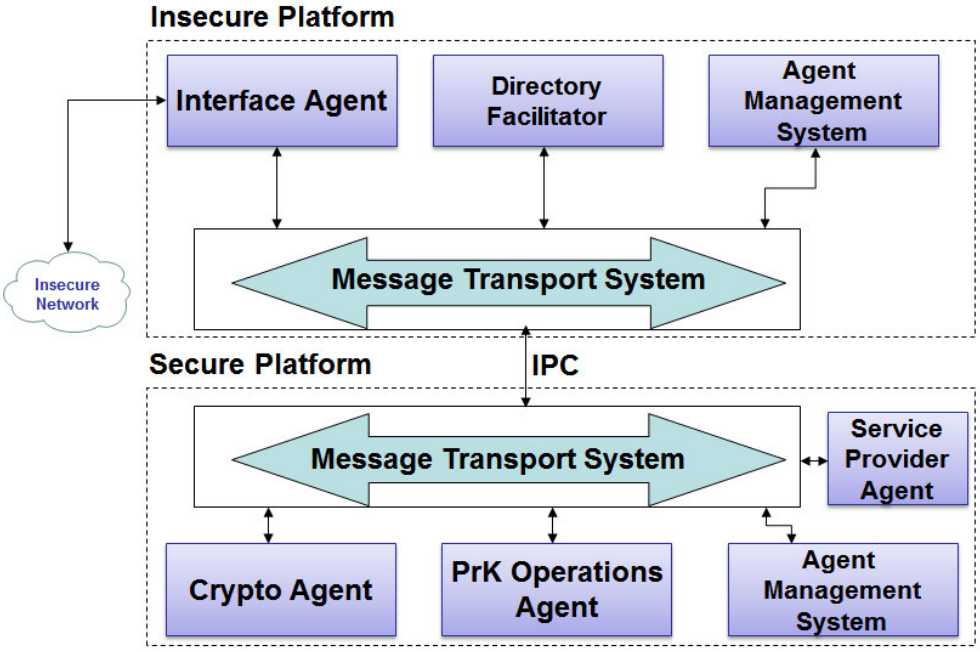

Figure 12 - The Proposed Architecture

3.1 The Insecure Platform

The insecure platform handles all network traffic, transfers most cryptographic operations to the secure platform and must run in a separate process and memory space.

In order to mitigate any damage caused by attacks, the insecure platform must not store session keys, private keys or user credentials.

Even though most services or daemons run on root or system accounts, the process in which the insecure platform runs should not be executed with such high privileges.

The Directory Facilitator exists to help interaction among other agents, such as agents performing the role of plugins.

3.1.1 The Interface Agent (IA)

The IA is an intelligent agent responsible for providing a transparent security layer for a service running in a server or a client application.

The IA should have the autonomy to discard non protocol compliant requests and block IP addresses that the IA interprets as sources of attacks for short or long periods at its discretion. For instance, the more an IP address is used to attack the system, the

longer it will be blocked. As a result, the Interface Agent will play the role of a NIDS (2.1.1 .

The IA should have a Knowledge Base to log attacks attempts and use those attacks as a reference for learning and for identification of future attacks.

The IA is the only point where services or applications can initiate communications and exchange information with the authentication system.

The IA does not have access to private keys, the session keys, or user credentials. Hence, a successful attack at the insecure platform and a dump of the entire memory of the process to a remote attacker would not reveal any sensitive information.

The IA will route encrypted data as well as unencrypted service requests to the Crypto Agent in the secure platform, after it is satisfied that the request is not an attempt to attack the system.

The IA may handle digital signature verifications, which only require the use of the Public Key of a digital certificate. Likewise, the IA handles certificate chains validations.

More than one Interface Agent may exist in the insecure platform, in order to handle large amounts of service requests. Nevertheless, they express selfish behavior. They are not to cooperate with any other Interface Agent, although they are required to honor other agents requests (agents performing the role of plugins), if they are not busy, and treat those requests as if they were requests from outside the platform.

The roles performed by the IA agents make the entire system more resilient to Denial of Service Attacks, since they have the autonomy to discard packets arriving from addresses they believe to be harmful packets to the system.

3.2 The Secure Platform

The secure platform handles the session keys, private keys, user credentials and is responsible to execute service requests. If necessary, it may run under a root or system account. It must also run as an independent process.

No foreign or visiting agents may exist in the secure platform. As a result, the Directory Facilitator module is not required.

3.2.1 The Private Key Operations Agent (POA)

The POA is a reactive agent whose only design goal is to respond to the Crypto Agent messages. It only performs the following specific functions with the Private Key:

- Asymmetric decryption with the Private Key
- Signing with the Private Key (digital signature generation)
- Generation of the Public and Private Key pair, when required by an application

The POA must have the autonomy to decide whether or not to honor requests. For example: if it receives a hash to sign with the Private Key, but the received hash is composed of a string filled with spaces, it should not honor the request and return an error message. Likewise, if it understands the message to use the Private Key is for crypta-

nalysis purposes or Private Key guessing, it should not return a digital signature but an error message instead.

In order to enhance security, the POA thread must be the only one with the Private Key stored in memory.

The purpose of a specific agent to deal with the Private Key is to diminish the possibility of Private Key theft or misuse. Also, the mathematical operations done with an asymmetric Private Key are at least one hundred times slower than an operation done with symmetric keys[44]. Having a dedicated agent to deal with Private Key operations, frees the Crypto Agent to perform other duties.

3.2.2 The Crypto Agent

The Crypto Agent is a reactive agent, who manages communications between the Interface Agents and the Service Provider Agents, even if the request comes unencrypted. There is not direct communications between an Interface Agent and a Service Provider Agent.

The Crypto Agent handles all symmetric session keys. If a session key has not been established yet, and a creation of a session key is necessary, the Crypto Agent will forward the request to the POA. The message returned by the POA will be the either the session key itself or part of the session key.

If the request comes after a secure session has been established, the Crypto Agent will decrypt the request, using the proper session key and authenticate the requester if required by the application. After the successful authentication, the Crypto Agent will forward the request to be processed by a Service Provider Agent. The returned results will be encrypted with the session key and returned to the IA, which will then forward the results to the client application.

The Crypto Agent also has the role of a dispatch center. It has a directory with the current status of each Service Provider Agent. It assigns requests to the first available Service Provider Agent. If all Service Provider Agents are busy, it may queue pending requests or return a too busy message to the IA. Developers are free to specify how many Service Provider Agents their solution will have and how their applications will deal with exhausted resources.

3.2.3 The Service Provider Agent (SPA)

The Service Provider Agent is a reactive agent responsible to process the requests and return the results to the Crypto Agent. As soon as it receives the request, it will send a message to the Crypto Agent, informing its status changed from “READY” to “BUSY”.

The SPA will use the authenticated user account or a specific visitors account to process to request. Is the access is denied, because the account does not have the necessary privileges, the SPA will inform the Crypto Agent that the user has insufficient privileges and that its status is now “READY”, meaning it is available to process other requests.

If the user is authorized, the SPA will process the request the way it was designed to do. When the request is processed, the SPA will return the results to the Crypto Agent and inform the Crypto Agent that its status is now “READY”.

4 Implementation

To demonstrate the feasibility of the proposed architecture, we implemented a prototype with two processes: the first emulates the Interface Agent of the insecure platform. The second emulates part of the Crypto Agent and Private Key Operations Agent of the secure platform. Each platform was modeled as an independent executable. The prototype was implemented with Microsoft Visual FoxPro 9.0 SP2. Communications between platforms is achieved by named pipes[45]. RSA[46] Asymmetric cryptographic functions are provided by Chilkat[VI] commercial libraries. The FoxPro language has the DISPLAY MEMORY [TO FILE] command, which dumps every variable, array and object defined into a text file. This mimics the Heartbleed Attack or a Buffer Overflow Attack that executes a memory dump into a remote location. An edit box and three command buttons emulates requests handled by the Interface Agent, while decryption with the Private Key, digital signature and key pair generation emulates the Private Key Operations Agent.

The secure platform do not allow short strings to be decrypted with the Private Key or a digital signature of a hash containing a single character duplicated several times to be digitally signed.

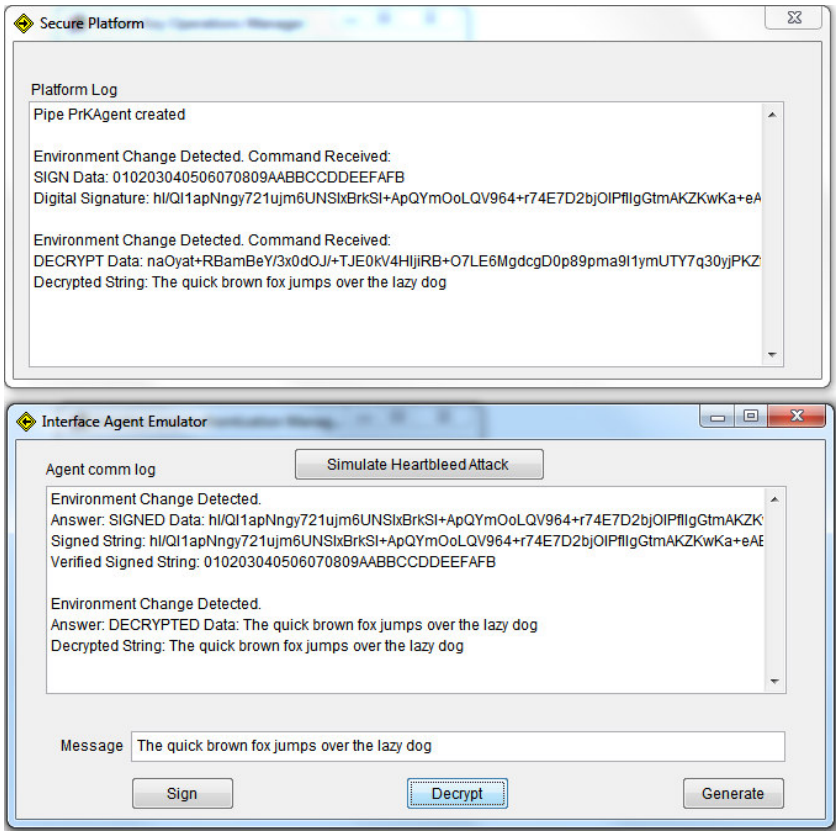

Figure 13 - Digital Signature and Asymmetric Decryption

Figure 13 displays two operations: first the digital signature of the emulated hash 0102030405060708090AABBCCDDEEFAFB and, second, an asymmetric decryption operation. The plain text "The quick brown fox jumps over the lazy dog" was encrypted with the Public Key before it was sent to the secure platform for decryption.

By clicking in the "Simulate HeartBleed Attack" button, we invoke a method which executes the command DISPLAY MEMORY TO FILE memorydump.txt and a instanti-

VI http://www.chilkat.com

ates a form object that opens the generated file, emulating a remote view of the defined variables of the insecure platform.

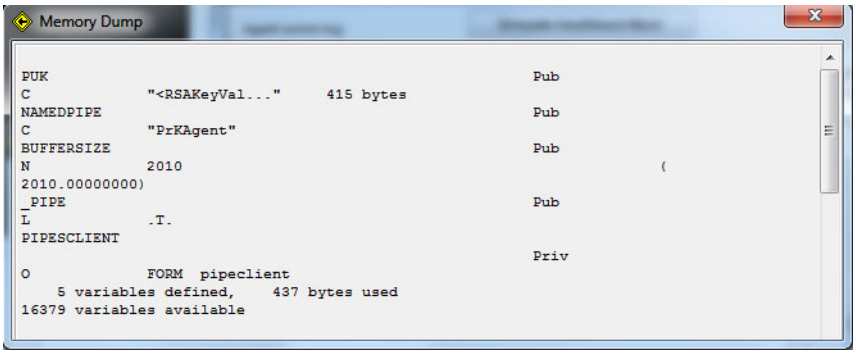

Figure 14 - Memory Dump Remotely Visualized

Even dumping the entire memory from the insecure platform, the Private Key was not revealed. Only the Public Key (PUK) was shown.

5 Related Work

Torrellas and Sheremetov (2003) [47], proposed an architecture for agent security and authentication. In their proposal, they used a CryptoAgent, responsible for encryption and decryption operations. They argued that the advantage of such modular design was that other components (even mobile agents) could use the functionality offered by CryptoAgent and that the platform could be made unaware of any additions or changes to cryptographic functions offered by this agent. Our proposal also takes in consideration the transparency of cryptographic methods offered by the Private Key Operations Agent, but our primary concern is the protection of sensitive data, particularly the Private Key, from other unauthorized agents or programs.

Likewise, Shakshuki et al. (2004) [48] proposed a Multi-Agent System to act as a middleware between users and the network. The middleware was specified to provide authentication, local and foreign authorization and service providing, where agents would be allocated to process the users' requests.

Shi et al. (2006) [49] proposed InfoShield: a security architecture designed to protect in memory information usage. InfoShield was designed to ensure that sensitive data are used only as needed by application semantics, therefore preventing misuse of information, by embedding specialized verification and tracking instructions inside the applications.

Lee at al. (2016) [50] proposed a two phases solution for computing environments for IoT (Internet of Things) services. The solution contains a secure compiler to identify and prevent weaknesses in the input program source code in C/C++, while the Secure Virtual Machine, monitors and handle Buffer Overflow Attacks and Exception Handlers.

6 Conclusions and Future Works

In this paper, we proposed an architecture capable of mitigating Buffer Overflow Attacks using Multi-Agent Systems concepts. We showed that, even if the insecure plat-

form is successfully exploited, sensitive data such as user credentials, session keys and, most importantly, the Private Key are not revealed. Because agents do not have a full perspective of the entire system, they are a very good approach to build applications and services in which security is paramount, since, from the security point of view, information is disclosed on a need to know basis. Agents also improve the resilience of the architecture to Denial of Service Attacks since agents can be dynamically created as needed or terminated by the Agent Management System, if they are not fulfilling their roles correctly. Also the Interface Agents are intelligent and capable of detecting attacks and block the originating addresses.

It would also be very difficult for an attacker to inject code while preserving the Agent Communications Language. Any ACL incompliant message would be immediately discarded by the Message Transport System, and the originating agent penalized. If an exploited agent insists on sending incompliant ACL messages it will eventually be terminated by the Agent Management System.

In the future, we hope to use this platform to model and implement authentication services and protocols.

A separate study must be done in order to assess performance issues, since it is likely the proposed architecture would run slower than a single service or daemon.

References

[1] IETF. RFC5246. **The Transport Layer Security (TLS) Protocol version 1.2**
Accessed on: 12/10/2013
Available at: http://tools.ietf.org/html/rfc5246

[2] Das, M. L., & Samdaria, N. (2014). **On the security of SSL/TLS-enabled applications**. *Applied Computing and Informatics, 10(1-2), 68–81*. http://doi.org/10.1016/j.aci.2014.02.001

[3] Xiao, P. et al. (2014). **An access authentication protocol for trusted handoff in wireless mesh networks**. *Computer Standards and Interfaces, 36(3), 480–488*. http://doi.org/10.1016/j.csi.2013.08.016

[4] Hsu, F; Guo, F; Chiueh, T. **Scalable Network-based Buffer Overflow Attack Detection**
Accessed on: 12/01/2014
Available at: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4579534

[5] Heartbleed
SCHNEIER, Bruce. Heratbleed
Accessed on: 04/10/2014
Available at: https://www.schneier.com/blog/archives/2014/04/heartbleed.html

[6] Mansfield-Devine, S. (2014). **Hacking on an industrial scale**, *Network Security, Volume 2014, Issue 9, September 2014, Pages 12-16*, ISSN 1353-4858, http://dx.doi.org/10.1016/S1353-4858(14)70092-3.

[7] WOOLDRIDGE, M. **An Introduction do Multiagents Systems**. John Wiley & Sons, LTD. 2002 pp-15-17

[8] SINGH, J et al. (2011). **Disclosure control in multi-domain publish/subscribe systems**
Accessed on 12/02/2014

Available at:
http://dl.acm.org/ft_gateway.cfm?id=2002283&ftid=993809&coll=DL&dl=ACM&CFID=60475
0411&CFTOKEN=20268464

[9] Ashish KUNDU, A; BERTINO, E. (2011). **A New Class of Buffer Overflow Attacks**
Accessed on: 05/28/2015
Available at: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=05961725

[10] FOSTER, J et al. (2005). **Buffer Overflow Attacks: Detect, Exploit, Prevent**
Syngress Publishing, Inc

[11] DAY, D at. Al. (2010). **Detecting Return-to-libc Buffer Overflow Attacks Using Network Intrusion Detection Systems** *Fourth International Conference on Digital Society*
Accessed on: 05/28/2015
Available at: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=05432802

[12] STRACKX, R et al. (2009). **Breaking the memory secrecy assumption**
Accessed on: 05/28/2015
Available at: http://dl.acm.org/ft_gateway.cfm?id=1519145

[13] LIANG ,Z; SEKAR, A. (2005) **Fast and automated generation of attack signatures a basis for building self-protecting servers**. *ACM/IEEE, 2005. Proceedings of the 12th ACM conference on Computer and communications security*. pp. 213-222.

[14] RATHOD, P et al. (2014) **A survey on Finite Automata Based Pattern Matching Techniques for Network Intrusion Detection System (NIDS)**
Accesed on: 06/17/2015
Available at: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7002456

[15] SADEGHI, A et al. (2015) **Securing Legacy Software against Real-World Code-Reuse Exploits: Utopia, Alchemy, or Possible Future?**
Accessed on: 06/16/2015
Available at: http://dl.acm.org/ft_gateway.cfm?id=2737090

[16] SHACHAM, H et al. (2004) **On the effectiveness of address-space randomization**
Accessed on: 06/17/2015
Available at: http://dl.acm.org/ft_gateway.cfm?id=1030124

[17] SERBA, F. (2012) **The info leak era on software exploitation**
Accessed on: 06/17/2015
Available at: https://media.blackhat.com/bh-us-
12/Briefings/Serna/BH_US_12_Serna_Leak_Era_Slides.pdf

[18] IETF. RFC6520. **Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS) Heartbeat Extension**
Accessed on: 06/01/2014
Available at: http://tools.ietf.org/html/rfc6520

[19] WILLIAMS, C. (2014). **Anatomy of OpenSSL's Heartbleed: Just four bytes trigger horror bug**
Accessed on 06/01/2014
Available at: http://www.theregister.co.uk/2014/04/09/heartbleed_explained/

[20] López, Fabiola López; **Social Power and Norms**. PhD. Dissertation University of Southampton, 2003.
Accessed on : 05/26/2016

Available at: http://www.cs.buap.mx/~fabiola/phdthesis.pdf

[21] Luck, M, et. al.; **Agent technology roadmap: Overview and consultation report**, 2005
Accessed on : 05/26/2016
Available at: http://www.agentlink.org/roadmap/roadmapreport.pdf

[22] Gerhard Weiss. **Multiagent Systems: a modern approach to distributed artificial intelligence**. Massachusetts Institute of Technology, 2001. 619p

[23] Bratman, M.E; **Intention, Plans, and Practical Reason**. Harvard University Press, Cambridge, MA (1987)

[24] Rao, A.S., Georgeff, M.P; **BDI-agents: from theory to practice**. In: Proceedings of the First Intl. Conference on Multiagent Systems. San Francisco (1995)

[25] Georgeff, M.P. et.al.; **The Belief-Desire-Intention Model of Agency**. In: Müller, J., Singh M.P., and Rao, A.S. (eds.): Proceedings of the 5th International Workshop on Intelligent Agents V : Agent Theories, Architectures, and Languages (ATAL- 98). Lecture Notes in Artificial Intelligence, Vol. 1555, pages 1–10. Springer Verlag, Hedelgerg, Germany (1999)

[26] Sen, S., Weiss, G; **"Learning in multiagent systems"** in Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence, G. Weiss, Ed. Cambridge, MA: MIT Press, 1999, ch. 6, pp. 259–298.

[27] Weiss, G., Sen, S; **Adaptation and Learning in Multiagent Systems**. Lecture Notes in Artificial Intelligence, Vol. 1042. Springer-Verlag, Berlin Heidelberg New York (1996)

[28] Stone, P., Veloso, M; M**ultiagent Systems: A Survey from a Machine Learning Perspective**. Autonomous Robotics, 8(3):345-383 (2000)

[29] Khalil, K. M., et al. **"MLIMAS: A Framework for Machine Learning in Interactive Multi-Agent Systems"** Procedia Computer Science 65 (2015): 827-835.

[30] Martinez-Gil, F, et. al. ; **Emergent Collective Behaviors in a Multi-agent Reinforcement Learning Pedestrian Simulation: A Case Study**. In Proceedings of Workshop on Multi-Agent Systems and Agent-Based Simulation (2014), 228-238

[31] Russell S., Norvig, P; **Artificial Intelligence: A modern Approach, 2nd Ed**. (2003) New Jersey: Prentice Hall

[32] Artz, D; Gil, Y; **A survey of trust in computer science and the Semantic Web**. Journal of Web Semantics: Science, Services and Agents on the World Wide Web (2007)

[33] Silva, V et al. (2007). **Governing Multi-Agent Systems**
Accessed on : 12/01/2014
Available at: http://link.springer.com/content/pdf/10.1007%2FBF03192407.pdf

[34] The Foundation for Intelligent Physical Agents. **FIPA ACL Message Structure Specification**
Accessed on: 11/28/2014
Available at: http://www.fipa.org/specs/fipa00061/SC00061G.pdf

[35] ASKDEFINE. **Etymology of the word cryptography**
Accessed on 06/18/2013.
Available at http://cryptography.askdefine.com/

[36] RUSSEL, Deborah; GANGEMI, G. T. Sr. **Computer Security Basics.** O'Reilly & Associates, 1991. p. 169-171

[37] SCHNEIER, Bruce. **Applied Cryptography 2nd edition.** John Wiley & Sons, 1996. p. 33

[38] SCHNEIER, Bruce. **Applied Cryptography 2nd edition.** John Wiley & Sons, 1996. p. 15-16

[39] STALLINGS, William. **Cryptography and Network Security Principles and Practice Fifth Edition.** New York: Prentice-Hall, 2011. p. 33-35

[40] DAHAB, R.; LÓPEZ-HERNÁNDEZ, J.C; **Técnicas criptográficas modernas: algoritmos e protocolos**. Instituto de Computação – UNICAMP 2007 p.32-37

[41] National Institute of Standards and Technology (NIST). (2012). Secure Hash Standard (SHS) (FIPS PUB 180-4).
Accessed on: 07/04/2014
Available at: http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf

[42] SUBRAMANYA, S.R.; YI Byung K. (2006) **Digital signatures**. IEEE March/April 2006
Accessed on 11/06/2013
Available at
http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=1649003&queryText%3DS.R.+SUBRAMANYA+AND+BYUNG+K.+YI

[43] PARK, B-K et al. (2013). **XpeedQ: A Reliable and Efficient Application Level Message Queue**
Accessed on: 06/18/2015
Available at: http://dl.acm.org/ft_gateway.cfm?id=2513276

[44] ROSEMBERG, M.R. (2014). **SRAP - A new Authentication Protocol for Semantic Web Applications**. MsC. Thesis. 2014. Pontifícia Universidade Católica do Rio de Janeiro, Dep. of Informatics, Rio de Janeiro. 72p Retrieved from
http://www.dbd.puc-rio.br/pergamum/tesesabertas/1221733_2014_completo.pdf

[45] MSDN. Pipe Reference.
Accessed on 11/12/14
Available at:
http://msdn.microsoft.com/en-us/library/windows/desktop/aa365784%28v=vs.85%29.aspx

[46] Patidar, R; Bhartiya, R. (2013). **Modified RSA Cryptosystem Based on Offline Storage and Prime Number**
Accessed on: 06/23/2015
Available at: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6724176

[47] TORRELLAS, G; SHEREMETOV, L. (2003). **An Authentication Protocol for Agent Platform Security Manager**.
Accessed on: 09/11/14
Available at: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1247764

[48] SHAKSHUKI, E et al. (2004). **Multi-agent System for Security Service**.
Accessed on: 09/11/14
Available at: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1283928

[49] SHI, W et al. (2006). **InfoShield: A Security Architecture for Protecting Information Usage in Memory**.

Accessed on: 05/28/2015
Available at: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=01598131

[50] Lee, Y. et al. (2016). **Design and implementation of the secure compiler and virtual machine for developing secure IoT services**. *Future Generation Computer Systems*. http://doi.org/10.1016/j.future.2016.03.014