Design and Implementation of Computer Worms Based on Monitoring Replication and Damage

Yazed B. Al-Saawy1† and Sulaiman Al amro2††

Islamic University in Madinah, Qassim University

Summary
This paper will present the formalisation stage of the development of the W DS. This formalization was achieved using CCA and was validated through simulation using ccaPL. The notation for CCA and the corresponding notation for ccaPL was presented. Moreover, this paper presented the formalised processes of each ambient as equations in CCA .Internal validation through simulation was achieved through a number of differ- ent scenarios which were designed to validate if the W DS was capable of detection and the sending and receiving of files and messages. Overall, the simulation was successful in showing that the modelling of the system had achieved the predefined requirements which included detection through signature-based and behaviour-based techniques, the W DS can now be implemented in JADE.

Key words:
Computer worms; behavior detection, replication; damage..

1. Introduction

This paper presents the implementation of the proposed worm detection system (W DS) of this study. The detection components of the system include an object handler, a database, hosts and a dummy host, each of which has its own class.

There is an explanation of the functioning of the W DS and how it operates within the M AS (Multi-Agent System). The W DS is designed to detect known and unknown worms using a dummy host as part of the detection process. This ter will describe how the implementation of the W DS is achieved using Java Agent Development Framework (JADE). Specifically, there is an explanation of the mapping between CCA and JADE where it will be shown that JADE is capable of implementing the functions of the W DS which include managing and controlling the sending and receiving of messages and files. Moreover, JADE can also implement the detection mechanisms.

The paper begins by presenting the main components of the W DS, it then moves into system design using class and sequence diagrams which illustrate the system structure and processes respectively. Thereafter, there is an explanation of the mapping between CCA and JADE, and finally, an explanation of the implementation of the system using the Java Agent development framework JADE.1.

2. Literature Review

2.1 Malware Mechanisms

El-Moussa and Jones [1] say that different types of malware show different behaviours. Malware often initiates actions of which the user is unaware, i.e., calling up an application or executive file [2]. A program may contain features the user requires but may also invoke unwanted behaviour. Thus, malware is able to enter a system with all the rights that are appropriate to the user or application but compromises data integrity [2]. Malware can function without being confined by a policy and/or deposit files that contain exploits for popular applications such as creation or viewing of documents and pictures [3]. These threats target exploits within the computer software to insert viral or malicious programs [4]. Until now systems have focused on preventing this exploitation, whereas, the present study uses this exploitation to prevent infection and damage to networks. Malware char- acteristics evolve with time yet often they fall into several categories of traits [5] wherein a malware is generally described by four attributes of its operation:

Propagation: refers to the mechanism that enables malware to be distributed to multiple systems [6]. Malware may perform vulnerability exploitation of server or client-side software; may be loaded by an intruder manually; may be autonomous and, may require user involvement such as launching an email attachment [5].

Malware's propagation mechanisms can be subtle and dangerous when it penetrates the infrastructure and spreads. Once malware has infected an infrastructure, there is generally some means by which it moves to other systems [7]. To achieve best replication results, different components of self-replication may also be dis- tributed among several processes that are communicating with each

Manuscript received November 5, 2016
Manuscript revised November 20, 2016

other, since its purpose is to infect as many hosts as possible [8]. Thus, malware can infect and spread over wired and wireless networks. By its ability to inter-operate between the Internet and wireless networks, along with improved functionalities and mobility, malware propagation in these networks has the potential to spread extremely fast [9], propagating to other nodes and compromising the network as malware takes advantage of certain weakness in network mobility [10].

Infection: refers to the installation routine used by the malware as well as its ability to remain installed despite disinfection attempts [6]. Malware may either run once or may maintain a persistent presence on the system; may assemble itself dynamically by downloading additional components or may attach itself to benign programs or functions as a stand-alone process [5].

Infection mechanisms, attack vectors and malware payloads continue to evolve as evidenced by the greater sophistication of exploits [11], such as malware links embedded in emails. Malware infection has gone from simply opportunistic to seem- ingly targeted attacks. These types of attack are often preceded by considerable open-source intelligence collection referred to as spear phising, during which specific individuals are sought to launch an attack against [7].

Self-defence: refers to the methods used by malware to conceal their presenc and resist analysis, such techniques may also be called as anti-reversing capabilities [6]. It may avoid signature-based detection by changing itself; may time its actions to take place during busy periods or to go slowly to avoid being noticed or may be designed to thwart analysis attempts, i.e., using a packer that encrypts the original executable and decrypting it at run time [5]. Attackers also circumvent the de- fence mechanism using obfuscation by altering the shape of data in order to avoid pattern-matching detection [12]. All too often, malware presents itself as armoured or obfuscated primarily to circumvent network security protection mechanisms [13].

Capabilities: refers to software functionality available to malware operator [6]. Malware may be designed to collect data by sniffing the network, recording keystrokes and screenshots, and locating sensitive files; may be programmed to wreck havoc on a system; and may provide the attacker with remote access to the system by acting as a backdoor [5].

Malware is capable of creating a persistence mechanism through the Registry autorun subkey, causing the program to start-up each time the system is rebooted [13]. However, Carvey [18] says not all persistence mechanisms reside within the Registry, as some, identified as file infector, infect executable or data files, enabling it to run whenever the executable file is launched or the data file is accessed.

Knapp [7] brings attention to a malware type called the advanced persistent threat (APT) which differentiates itself from other normal malware by its capability to shift from broad, untargeted attacks to more directed attacks that focus on determining specifics about its target network and attempts to remain hidden and proliferate within a network, leading to the persistence of the threat.

2.2 Computational Model

The design of the detection system is modelled using ambients which will be formally specified using CCA. Therefore, the computational model reflects a system modelled using ambients. Here we describe the components of the detection system and demonstrate where they are located in the overall architecture, followed by a presentation of the computational model using CCA.

The architecture of the W DS is presented in Figure 1. The architecture comprises of components of the overall networked system of computers which the W DS is designed to protect, these are represented as ambients and include the department and the hosts within that department. The architecture is also comprised of components of the W DS itself, also represented as ambients, which include the Dummy Host (DH), the Object Handler (OH) and the database (DB). For the purposes of this study we are concerned with the architecture, however, the system would work with multiple departments.

It is important that the computational model allows both the architecture and the interactions between the system components to be understood . For example, a host within a department becomes infected with a worm which has the ability to move from one host to another via the network. Located in the network between the various hosts is the Object Handler (OH) which has two main functions which are firstly, to manage the transfer of files and messages and secondly, is comprised of part of the overall detection mechanism, i.e., detection by signature.

The following is a brief overview of the components of the system

• Hosts: The hosts are the machines.

• Dummy Host: This is the host that allows itself to be infected with worms so that they may be detected through behaviour which includes damage and replication.

• Object Handler: This is the developed middleware of the present study, which has two functions - detection of worms using signature-based detection and management of files and messages.

• Database: The database contains a list of known worms against which the Object Handler checks the file, this forms part of the overall detection mech- anism.

• win32: The win32 ambient is a folder which contains the executable and system files, each of which are also considered as ambients. These files include system files such as cdrom.sys and executable files such as cmd.exe. The win32 ambient is the location in the Dummy Host ambient where worms are allowed to carry out replication behaviour and damages.

As shown in Figure 1 the W DS is comprised of the Hosts, the Dummy Host DH, Object Handler OH and Database DB . Before the system can be formalised it is necessary to model the W DS whereby the system's components are represented as ambients. Now we will illustrate this modelling and also demonstrate how the subsequent model will be translated into a formal specification using CCA.

At this point the visual model provides a high level of abstraction of the W DS and illustrates the components as ambients, the overall architecture and the process relationships between the ambients. The next stage is to formalise the ambients, the W DS architecture and the system processes, which includes the detection mecha- nism, in CCA.

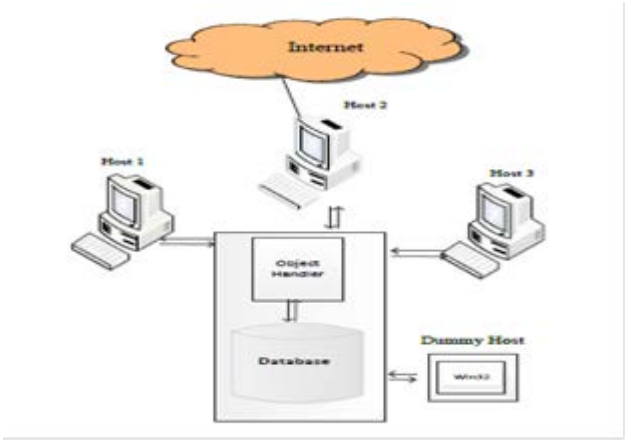

Fig. 1 Architecture of the Worm Detection System

Now that we have the visual modelling as an architecture of components as shown in Figure 3.7, and before we move to formalising the WDS in CCA, it is first necessary to show the corresponding visual representation of the components of the system as ambients, this is illustrated in Figure 3.8. There is a one-to-one correspondence between the visual modelling shown in Figure 1 and the modelling as ambients in Figure 2. The following describes the correspondence between the two graphics.

• Host 1 and Host 3 as components in the architecture of the system are the networked machines (see Figure 1) that send and receives files and messages, which the WDS is designed to protect from worms. In the graphical modelling using ambients Host 1 and Host 3 are denoted as H1 and H3 respectively (see Figure 2).

• As a component shown in the architecture of the system, the Dummy Host is a machine or agent (see Figure 3.7). In the corresponding equivalent in the graphical modelling using ambients the Dummy Host is denoted as DH and is an ambient that contains the child ambient win32 and is a sibling ambient to other system components (see Figure 2).

• Another component of the W DS represented in the architecture is the Object Handler which manages the sending and receiving of files and messages, this component is located between the networked Hosts (see Figure 3.7). In the graphical modelling the Object Handler is an ambient denoted by OH and is a sibling ambient to H 1, H 3 and DB (see Figure 2).

• The Database component of the system, which contains a list of signatures of known worms, is separate from the other components (see Figure 3.7). This is clearly illustrated in the graphical modelling where the Database is denote as the ambient DB and is a sibling to the other ambients (see Figure 2).

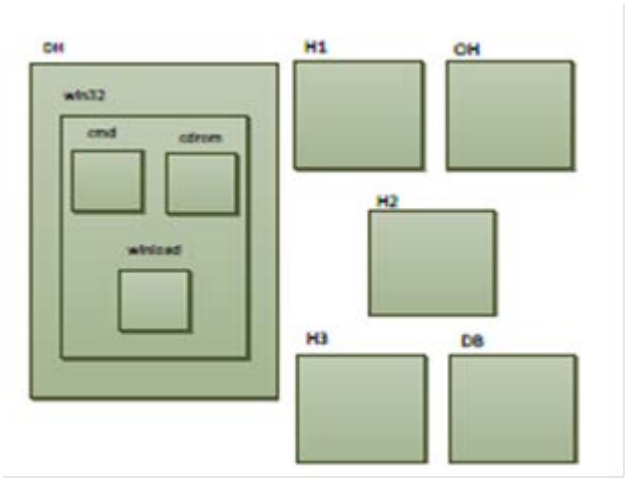

Fig. 2 Graphical Representation WDS

Below is a textual representation of the above WDS using CCA:

DH [win32[cdrom[Pcdrom]| cmd[Pcmd]| winload[Pwinload] | Pwin32] | Pdh]
| H1 [PH1]
| H2 [PH2]
| H3 [PH3]
| OH [POH]
DB [PDB]

2.2 Detection of Worm Behaviour

This section provides an overview of the detection mechanism of the W DS. The flow diagram in Figure 3.9 shows the mechanism for worm detection, here the process shows any order of replication or damage respectively. Firstly, we consider the mechanism where replication is detected. If replication is detected, then this indicates the presence of a worm, the system will then check for damage, it is important to note that checking for damage is for information purposes only, as detection has already taken place through detection of replication behaviour. Knowing the type of damage caused by a worm will indicate the type of worm. If replication is not detected then the system will check for damage, if damage is detected then the result is that malware has been detected which could include a worm, the reason for this is because damage without detecting replication means that it is probable that malware other than a worm is detected. There are five specific types of damage corresponding to each of the five worms detected which are corrupts file, destructive file, create files, change file size and copy files. Damage can also be detected through the hash directory. If no damage is detected then the result will be 'clean'. The processes that have been described here are implemented into an algorithm which shows all detection possibilities. This algorithm is shown in Table 3.1 where D is the Boolean variable representing damage, R the Boolean variable representing replication, the results are either clean or infected.

Table 1: Infection Matrix

D / ***R***	***Damage***	***No damage***
Replication	Infected	Infected
No replication	Infected	Clean

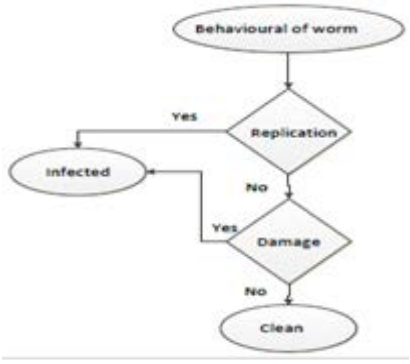

Fig. 3 Detection of Behaviour

3. Components of Detection System in JADE

The WDS comprises a number of different components; each component has a specific task and includes the Hosts, the Dummy Host, the Object Handler and Database. These components are implemented in the system using JADE . Besides the components of the system, JADE is also used to implement replication behaviour and damage derived from the classification. Figure 4 is a graphical representation of the components of the system.ables, Figures and Equations

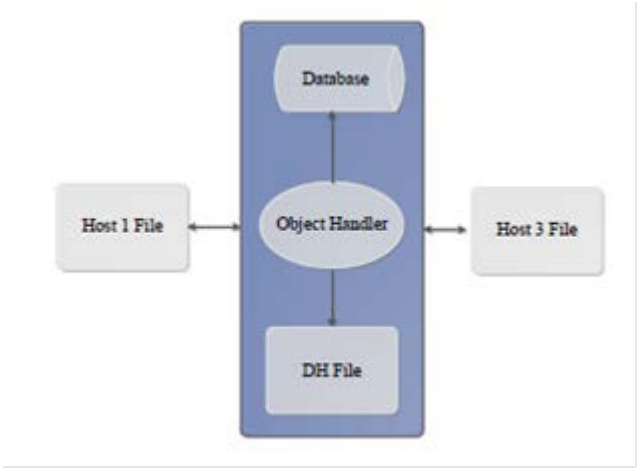

Fig. 4 System Components

• Host 1:- Host 1, shown in Figure 5.2, is written as Host1File in JADE as a class. Host 1 is one of the networked machines which the W DS is designed to protect and is capable of sending or receiving files or messages. For the expertimentation of the W DS it is the sending agent. When Host 1 engages in an action, e.g., sending file number 1, it is denoted in JADE as Host1File1 which can be seen in the JADE GUI. The reason it is necessary to include the name of the file is for communication purposes with the other components such as the Object Handler and other hosts, for example, the sending host is denoted in JADE as Host1File1, the receiving host, DH , has to be denoted as DHFile

1, if the name of the file being sent does not match, then communication will not take place.

• Dummy Host:- Dummy Host (DH) shown in Figure 5 is written as DH in JADE as a class. The DH as an agent always sends and receives files and is located in the system and is the intermediary agent between the sending and receiving agents (hosts). If a message or file is sent between two hosts it must first go to the DH. The DH is a machine that has permission to receive any files and messages without any authorisation required. Thus, if a file contains a worm it will be allowed to carry out replication

behaviour or damage in the DH and it is the observation and identification of this replication behaviour or damage which reveals the presence of a worm and the type of damage caused. As with Host 1, DH will be denoted in JADE as DHFile when engaged in sending or receiving a file, this is also shown in the JADE GUI interface.

• Host 3:- Host 3 is an agent of the system that sends and receives files, however, in most of the experimentation Host 3 is the receiving agent that will receive the sent file or message from the sending agent (Host 1). As part of the function of the detection system, it is important to note that the file or message will only be received by the destination host if no worm is detected. Thus, Host 3 run as a receiver is denoted by JADE as, for example, Host3File.

• Object Handler: - The function of the Object Handler is to manage the sending and receiving of files and messages, including objects, between the networked hosts (Agents) using a cyclical process which is a method of JADE whereby within JADE messages or data are sent and received repeatedly on a cyclical basis. The Object Handler is also involved in managing the signature- based detection in the database.

• Database:- The Database contains a list of existing worms and their signatures which is used as part of the first layer of the detection process, signature- based detection. The list of worms was gained from the centralised database available online. Moreover, the database is updated with the names of newly-detected worms and the location and type of damage caused. The database log will be updated every time detection of damage takes place.

4. System Design

This section presents the design of the W DS. Class diagrams and sequence diagrams are used to illustrate the design of the system. These two types of modelling

were adopted because the class diagram provides a visualisation of the specifica- tion requirements and system properties and the need of different objects, and the sequence diagram provides a sequential visualisation of how messages and files are sent and received. The system was designed using a high level specification duri4.1 Class Diagram ng formalization.

4.1 Class Diagram

The class diagram in Figure 5.2 shows the various classes (ambients) of the system. In the previous research [] it was shown how the ambients of the system were modelled and formally specified in CCA, this specification was necessary for implementation using JADE because it showed the components of the system, how they should be situated in relation to each other and the actions that they need to perform.

The system has networked hosts (which are Agents in JADE) namely Hosts 1 and Host 3 are instances of the Host class, additional hosts may be added which will also be instances of the Host class. They have a relationship with the Object Handler and the JADE GUI. These classes represent the networked machines of the system and are engaged in sending and receiving files and messages as well as to deny or clean file messages from the OH class.

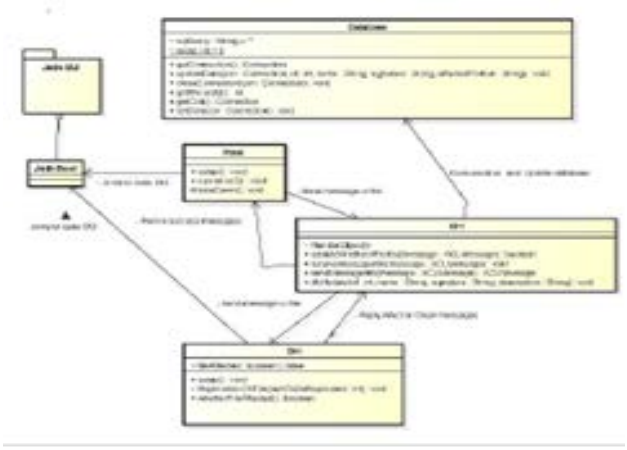

Fig. 5 Class Diagram

The OH class is an intermediary, or middleware, situated between the Host classes and the Database class as well as between the host classes and the DH (De- tection Handler) class. The main function of the OH class is to manage the send- ing and receiving of files and messages between the Host classes and the detection classes, i.e., the Database class and the DH class. The OH class has a controlling function and will allow or deny files and messages based on whether a file is clean or infected. It is the only class in the system that has a direct interaction with all the other classes of the system. Specifically, this class includes the identifyWhether- Affected which shows the operation of identifying if a worm has been detected as a type of Boolean and receiveMessageInfo which is the receiving of the message from the Dummy Host. The sending and receiving of messages is achieved through ACL messaging.

The Database class contains the signatures of known worms and is used in signature-based detection. This class only has a direct relationship with the OH and the DH classes. The OH sends files and message to the Database class for signature-based detection, and in order for the OH to open communication with the database a connection is

made. The file and the results of the signature-based detection are sent to the OH by the Database class. The Database class also receives information about newly detected worms including file name and signature which are updated in the database.

The DH class is the main detection component of the system and is responsible for detection through replication behaviour and damage. It has a direct relationship with the OH and the Database classes. The specific function of the DH class is to detect a worm through hashing for both replication and damage. The DH class contains the Hash directories and compares the hash values of the original file with the current file in order to determine if there is a change in these values. For this it uses the MessageDigest which compares the bytesToHex of the original file with the received file and decides whether or not the latter is clean and if it is clean, it is sent to the receiving host. If the file is infected it will inform the OH class and it will also update the database with the file information such as signature, name of file and type of damage

4.2 Sequence Diagram

A sequence diagram illustrates sequentially the interactions between the components of the detection system, i.e., the sending and receiving of files and messages and detection. More specifically, it shows the specific sequence of events when a file is sent from one host to another, which includes the messages that are sent and received as part of this process. A sequence diagram can be generic, where it shows all possible scenarios, or it can be in the instance form where it focuses on a specific scenario.

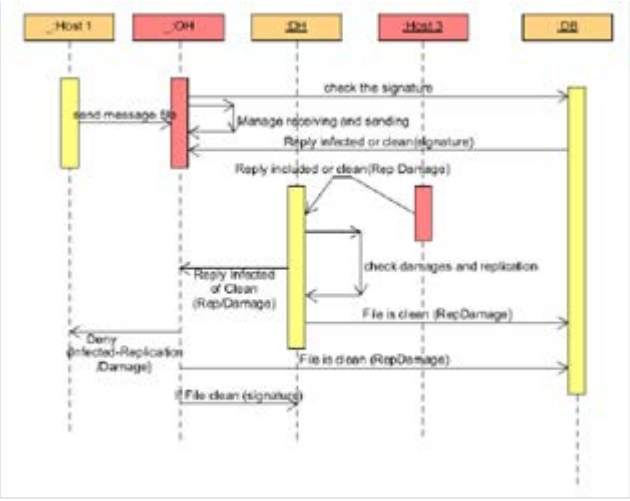

Fig. 6 Sequence Diagram

The sequence diagram above shows the sequential processes for the detection system and is considered generic because it shows the communication processes between the various classes, such processes are required for the implementation in JADE. These processes have already been established through modelling and formal specification using CCA, where the possible interactions between ambients, e.g., communication, were established and formally specified.

In the sequence diagram in Figure 6 the events that take place for both an infected and a clean file are represented. Here Host 1 wants to send a message to Host 3. The first event is that Host 1 sends a message or file to the OH, indicated by event 1. The OH checks the signature of the file or message with the database (DB) shown by event 1.2. After the signature-based detection has taken place in the database a reply message is returned to the OH indicating whether the file is infected or clean, shown by event 1.3. Thereafter, if the file or message is infected the OH will send a deny message to the sending host (Host 1) that the file or message is infected, shown by event 1.4. If the reply from the Database class says that the file is clean the OH will send the file to the DH for detection by replication behaviour and damage, shown by event 6. The DH will carry out detection indicated by event 2. Depending on whether or not detection takes place a reply that the file is infected or clean will be sent back to the OH, shown by event 3. If the file is infected the OH will send a deny message to the sending host (Host 1) that the file is denied and cannot be sent, shown by event 3.1. At the same time the DH will update the database with the infected file shown by event 4. In the case that no infection is detected in the DH, the OH will send the file or message to the receiving host, in this case Host 3, shown by event 5.

5. Mapping from CCA to JADE

Once the W DS has been modelled and formally specified in CCA the next step is to implement the system in JADE which is a software platform for developing multi-agent systems. During the imple- mentation it is necessary to consider the components and processes of the system, as described in the above, as overall requirements. More specifically, we need to consider the required system capabilities, the location of the various components or ambients, the required processes and the Dummy Host as part of the mechanism of detection, these considerations are illustrated in Figure 5.4.

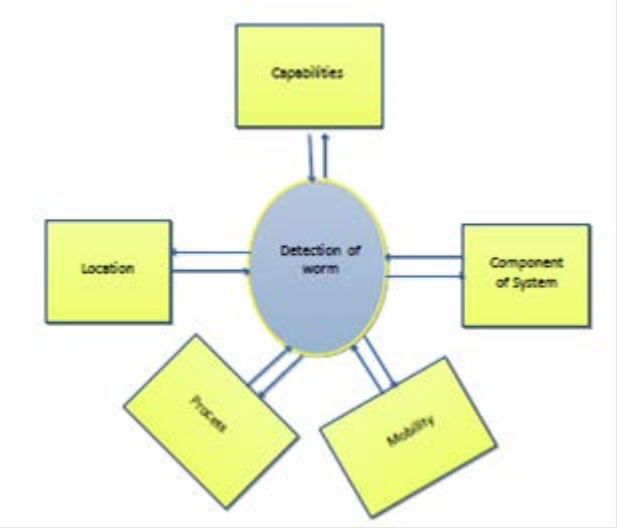

Fig. 4 Mapping between CCA and JADE

The translation between the system modelled in CCA and the system to be im- plemented in JADE is simplified by the fact that there is corresponding mapping between CCA and JADE, for example, CCA has ambients and JADE has agents, and both consider capabilities, location, processes and mobility and also both con sider communication. The implementation in JADE is achieved through mapping between CCA and JADE, which is presented in the table 2.

In the table above there is a clear mapping between the components and processes specified in CCA and their implementation in JADE. In reference to processes, the table shows how JADE implements systems behaviours, specifically, cyclical behaviour. The mobility attribute specified in CCA is implemented in JADE using migration. Importantly, the table also shows how communication is implemented, in this case the hand shaking process formalised in CCA is implemented in JADE through the data communicating properly with the RMI in Jade. This will involve the agent joining the RMA (Remote Monitoring Agent) in Jade using the ACL messages which contain send and receive messages to and from the Agents.

Table 2: Mapping from CCA to JADE

CCA	JADE
Ambient	Agent
Ambient location	Agent location
Parallel composition of processes	Parallel composition of threads.
Hand-shake message passing	Agent Communication Language (ACL).
Ambient mobility	Agent migration as agent mobility.

6. Implementation of WDS using JADE

JADE is used for the implementation of the detection system and is used in the worm detection mechanism. This section will describe this implementation in detail and includes the implementation of the components and the processes such as communication and update functions.

Once the database has been identified by JADE and the agents (Host 1, DH, Host 3) have been activated, they are ready for communication and sending and receiving files. The system is now ready to function and perform sending and receiving files and messages, detection and updating signatures and type of damage in the database. The system will not be terminated while the JADE GUI is running, this avoids the system terminating if one of the Agents stops working properly. Agents will automatically continue sending and receiving messages and remain in an active state unless there is a problem with the agents, such as a network communication error, this is one of the advantages of JADE. This sequence of actions of the system is illustrated in Figure 5. These sequential processes can also be found in the sequential diagram in Figure 3, for example, JADE System in Figure 3 is equivalent to ‘Start Jade.Boot’ in Figure 5, ‘parsing of message done in Object Handler’ in Figure 3 is equivalent to Object Handler manages messages in Figure

5 and finally, the ‘if file good, send (bad not send)’sequence in Figure 3 has its equivalent in Figure 5 as ‘Reply to sending agent deny or accept message’.

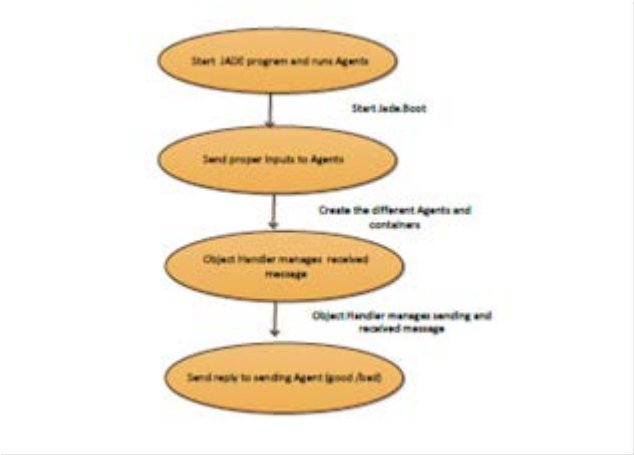

Figure 5: Sequence of System Actions

6.1 Implementation of Components of the System

Before the sending and receiving of files and the subsequent detection can take place the system has to be implemented. This section will describe the mechanisms be- hind the execution of the components of the system.

The first action required in running the system is to run the command prompt mysqld in order to run the service of the mysql to open a connection with the database. The database connection is established by connecting the MySQL of the database to JADE so that the database can be updated after detection takes place. To run the MySQLd, the J DBC mysql driver is required to provide a connection with the database as shown in listing 1.

Listing 1: Database Connections

```
1 publicConnection getConnection() {
2   try{Class.forName("com.mysql.jdbc.Driver").newInstance();
3 con= DriverManager.getConnection("jdbc:mysql:
4 //localhost/Wormdb1","root","123");
5 if(con!= null){
6 System.out.println("connections are correct");
7 setCon(con);}}}}}
```

The second action required in running the system is to run the command prompt Java jade.Boot - GUI in order to show all the agents in the platform via the JADE GUI interface, for example, DHFile1 (as receiver) and Host1File1 (as sender) in the Agent platform shown in the interface. In order for the agents to be identified, they first have been executed, this includes the Dummy Host because it is considered as an agent.The JADE interface GUI provides a visual representation of the Agents, Agent communication and their respective IP addresses, see Figure 6.

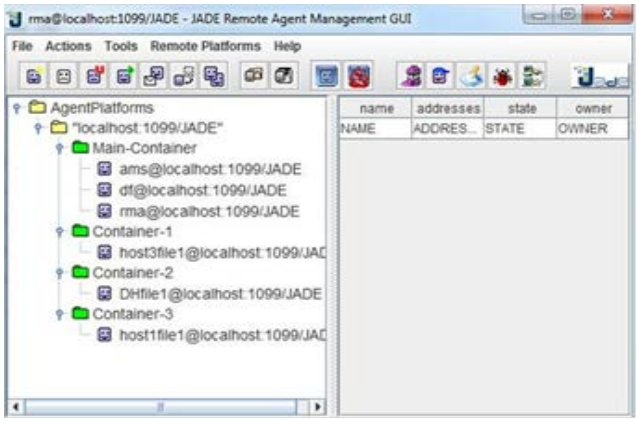

Fig. 6 Communication of All Agents

These hosts (Agents) are able to communicate with each other through Peer to Peer communication or wireless communication, the latter is verified by the ability to ping each other using the JADE Framework. An example of this is shown in the GUI in Figure 5.6. Each Agent can be executed as an Agent in a multi-agent system. For a detailed explanation of the multi-agent system using JADE. Moreover, the JADE framework allows the agents to be joined together within the system. This allows the communication between agents to take place through the JADE framework.

6.2 Structure of Detection Component

The detection component is comprised of two sub-components, namely; Object Handler for sending and receiving objects (files and messages) and managing signature- based detection and the Dummy Host for allowing replication behaviour and damage to take place for detection purposes. This section will describe how these sub- components are implemented in JADE.

6.2.1 Sending and Receiving Objects - Object Handler

Firstly, in order for a message to be sent it needs to have a destination, in this case the address of the receiving agent. Therefore, the Object Handler, as part of its role in sending and receiving messages, will receive a request from the sending Agent (host), and at this point the Object Handler will add the receiver's address. The sending agent will be given the specific address of the receiving agent (host), this will confirm where the message will be sent.

```
msgRx.addReceiver(new AID("Host3File1",
AID.ISLOCALNAME));)
msgRx.addReceiver( msg.getSender() );)
```

The Object Handler will add the receiver information. Here the sender of this information can be DH or Host 3 because they are the recipients of the file.

```
sendMessageInfo(ACLMessage message)
```

The Object Handler acts as a receiving agent from Host 1 and therefore, needs to add the sender's (Host 1) address, it is also the sender for receiving agent, Host 3 and adds this address as described above. This means that the reply from the receiving agent (Host 3) can simply be 'to sender' because the address has already been added by the Object Handler. This would be the case for any final destination receiver of the message in a multi-agent system.

The Object Handler thus handles the sending and receiving of objects, which includes files and messages. The following are the actions of the Object Handler.

1. identifyWhetherAffected(AclMessage):- This indicates if the file is infected or not. For this it will call the Hashing function and check the corresponding flag to see if it is infected with a worm.

2. receiveMessageInfo(ACLMessage message):- This method receives the message and verifies its content. This receives the message and replicates the information and creates corresponding files in the system and logs the message in the Object Handler.

3. sendMessageInfo(ACLMessage message):- This will be used to pass the mes- sage to the sender. This sends messages had been sent by the Message Object which is the file set in that object.

The code for this process is shown in the fo 2. The initial portion is the declarations. There is no main function for the Object Handler because it is the intermediary object between the agents of the system.
In addition to handling the sending and receiving objects generally, in the case that the detection component of the system detects a worm, the Object Handler will identify the deny message from the detection component and reply back to the sender that the file or message cannot be sent to the final destination.

Listing 2: Object Handler

```
if (conversationId4.contains("HostiFile1")) {
System.out.println(" this is called from the DptDH2File2");
if (msgRx!=null)
System.out.println( "== Answer" + " <- "
+  msgRx.getContent() + " from "
+  msgRx.getSender().getName() );
public ACLMessage sendMessageInfo(ACLMessage message) {
 ACLMessage msg = message;
String conversationId4 = msg.getConversationId();
  ACLMessage msgRx = new ACLMessage(ACLMessage.INFORM);
  String dataToParse = message.getContent();
```

Listing 3: Object Handler1

```
int firstPipeIndex = dataToParse.indexOf("|");
  int lastPipeIndex = dataToParse.lastIndexOf("|");
  if (dataToParse != null) {
  if (dataToParse.contains("|")) {
  data[0] = dataToParse.substring(0,firstPipeIndex);
  data[1] = dataToParse.substring(firstPipeIndex+1,
  lastPipeIndex);
  data[2] = dataToParse.substring(lastPipeIndex+1);
  replicationToBeDone = Integer.parseInt(data[2]);}
if (conversationId.equalsIgnoreCase("DptHost1File1")) {
    if (dataToParse != null)
if (data[1].equalsIgnoreCase("2"))
File1.createDestructiveFile();
msgRx.addReceiver(new AID("DH2File1",
 AID.ISLOCALNAME));
      try {
        msgRx.setContentObject(File1);
      } catch (IOException e) {
        e.printStackTrace() }
        return msgRx; }
```

In this case, the information passed by the command line arguments and the parsing of that information is sent to dataToParse. From this the corresponding arguments are gained and those arguments are verified against the corresponding arguments of the file or message.

6.2.2 Dummy Host

The second component is the Dummy Host where files and messages are sent for detection purposes, if the file or messages contains a worm, it will be allowed to cause damage. The cmd.exe is the executable command in the MS DOS prompt and it will be in the win32 directory. The way that the Dummy Host detects the damage is by matching. For example, the information for this cmd.exe is stored in a temporary folder in the system and the original file is matched against the same file type after infection, which will reveal a difference in the file size, which indicates the presence of a worm. The code for this detection is given below in listing 4.

Listing 4: Dummy Host detection by Damage

```
MessageDigest md = MessageDigest.getInstance("256");
  File newFile =  new File(fileName);
  FileInputStream fin = new FileInputStream(newFile);
   * To create DataInputStream object, use
   * DataInputStream(InputStream in) constructor.
  DataInputStream din = new DataInputStream(fin);
  byte b[] = new byte[(int) newFile.length()];
  System.out.println("   "+bytesToHex(output));
  String originalFileHash = bytesToHex(output);
  newFile =   new File(fileName);
  fin = new FileInputStream(newFile);
   * To create DataInputStream object, use
   * DataInputStream(InputStream in) constructor.
  din = new DataInputStream(fin);
  b = new byte[(int) newFile.length()];
  din.read(b); din.close();
  fin.close(); md.update(b);
  byte[] output1 = md.digest();
  System.out.println("   "+bytesToHex(output1));
  String currentFileHash = bytesToHex(output1);
  if (currentFileHash.equalsIgnoreCase(originalFileHash))
    System.out.println(" File Had not changed ");
  else {
System.out.println(" File Has changed ");
System.out.println(" Original File Hash value " + originalFileHash);
System.out.println(" Current file Hex value " + currentFileHash);
    changed = true;}
```

Here as per the logical data variations of the file, the affected file will be as output. Here is the original file hex code is calculated, then the hex code for the received file is calculated and then the Hash values of the original file and current file are compared, if there is a difference in these Hash values then damage has occurred.
The WDS is able to detect both known and unknown worms through detecting replication behaviour and damage. Specifically, there are different types of replication behaviour and damage, these are specified in

JADE. Although the detection of a known worm may take place by detecting both replication and damage and the detection of an unknown worm may be through damage only, the detection mechanisms are still the same for detecting both types of worm.

6.2.3 Updating Database

The method for inserting information about worms and updating the database involves firstly, inserting the worm information in the database referred to as worm info which contains id, name, signature and location Of Damage as illustrated in listing 5. This occurs after detection has taken place. Specified records can also be deleted from the database.

Listing 5: Database Connections

```
System.out.println("Database Connection successfully made");
Statement stat = (Statement) con.createStatement();
String sqlQuery = "Insert into wormsInfo (id, name, signature,
 locationOfDamage) values ("
  + getRecordId() + ",'" + name + "',  '" + signature + "','" +
  affectedPortion + "');";
System.out.println(" the SqlQuery " + sqlQuery);
stat.execute(sqlQuery);
sqlQuery = "SELECT count(*) " +
  " FROM wormsignature where wormName = " + "'" + name + "'";
ResultSet rs = stat.executeQuery(sqlQuery);
```

Listing 6: Dummy Host Detection by Replication

```
Void replicationOfFile() {
  for (replicationType = 3; replicationType <= 3; replicationType++) {
  System.out.println(" the value of the Replication Type " +
  replicationType);
    for (int i = 0; i < numToBeReplicated; i++) {
     if (replicationType == 1) {
     String fileOutputName = fileName;
     File outputReplicatedFile = new File(fileOutputName);
     if (!outputReplicatedFile.exists())
      outputReplicatedFile.createNewFile();
       FileOutputStream fw = new FileOutputStream(
       outputReplicatedFile.getAbsoluteFile(), true);
       } else if (replicationType == 2) {
       String fileOutputName = fileName.substring
       (fileName.lastIndexOf("\\")+1);
         directoryName = getParentFile();
         directoryName = directoryName + integer.tostring(i);
         fileOutputName = directoryName + "\\" + fileOutputName;
         writer.write("\n" + fileOutputName + 'Written');
         File outputReplicatedFile = new File(fileOutputName);
         boolean createFileDir = new File(directoryName).mkdir();
         if (!outputReplicatedFile.exists())
           outputReplicatedFile.createNewFile();
         FileOutputStream fw = new FileOutputStream(
             outputReplicatedFile.getAbsoluteFile(), true);
       } else if (replicationType == 3)
       {String directoryName = getParentFile();
       String fileOutputName = fileName.substring(fileName.
       lastIndexOf("\\")+1);
       String parsedFile = fileOutputName.substring(0,
       fileOutputName.indexOf("."));
         parsedFile = parsedFile + Integer.toString(i);
         parsedFile = parsedFile + '.'
         + fileOutputName.substring(fileOutputName
         .indexOf(".") + 1);
         parsedFile = directoryName + "\\" + parsedFile;
         writer.write("\n" + parsedFile + " Written");
         File outputReplicatedFile = new File(parsedFile);
         if (!outputReplicatedFile.exists())
           outputReplicatedFile.createNewFile();
         FileOutputStream fw = new FileOutputStream(
       outputReplicatedFile.getAbsoluteFile(), true);
       }
```

Secondly, once the aforementioned information about a worm has been inserted, the database is updated. When the system is inactive, the connections with the database will be closed.

Listing 7: Update Database

```
public void dbUpdate(int id, String name, String signature,
String description) {
    DBConnections dbConn = new DBConnections();
    Connection con = dbConn.getConnection();
    dbConn.updateData(con,
    id, name, signature, description);}

  public void closeConnection(Connection con) {
    try {
      if (con != null)
        con.close();
    } catch(Exception e){
      e.printStackTrace();}
```

Conclusion and Future Research

In summary, this paper has presented the implementation for the WDS using JADE. More specifically, it has shown how this implementation first required knowing the relationships between the components of the system using class and sequence diagrams. It has described the components of the system as well as de- scribing how these components are implemented in JADE through mapping. The paper followed the process sequentially from describing the components to be implemented in JADE, establishing the functions of each component using class and sequence diagrams and mapping components formalised in CCA with JADE. The future research will be the experimentation and evaluation of the WDS.

References

[1] F. El-moussa and A. Jones. Malware analysis: The art of detecting malicious activities. In Proceedings of the 7th European Conference on Information War- fare, page 51. Academic Conferences Limited, 2008.

[2] S. Furnell and J. Ward. Malware: An evolving threat. Digital crime and forensic science in cyberspace, pages 27–29, 2006.

[3] R. Sekar. Information flow containment: a practical basis for malware defense.In Data and Applications Security and Privacy XXV, pages 1–3. Springer, 2011.

[4] D. Harley and R. Vibert. AVIEN Malware Defense guide for the Enterprise. Syngress Media Incorporated, 2007.

[5] L. Zeltser. Analyzing malicious software. In CyberForensics, pages 59–83. Springer, 2010.

[6] BITS. Malware risks and mitigation report. Technical report, A DIVISION OF THE FINANCIAL SERVICES ROUNDTABLE, 2011.

[7] H. Carvey. Windows Forensic Analysis Toolkit: Advanced Analysis Techniques for Windows 7. Syngress Publishing, 2012.

[8] A. Volynkin. Advanced methods for detection of malicious software. ProQuest, 2007.

[9] K. Ramachandran. Stochastic and epidemiological models for performance evaluation of peer-to-peer networks. ProQuest, 2007.
[10] P. DE. Data Dissemination Protocol in Wireless Sensor Networks: Design, Modeling and Security. PhD thesis, The University of Texas at Arlington., 2008.
[11] E. Knapp. Industrial Network Security: Securing Critical Infrastructure Net- works for Smart Grid, SCADA, and Other Industrial Control Systems. Syn- gress, 2011.
[12] Y. Choi, T. Kim, S. Choi, and C. Lee. Automatic detection for javascript obfuscation attacks in web pages through string pattern analysis. In Future Generation Information Technology, pages 160–172. Springer, 2009.
[13] C. H. Malin, E. Casey, and J. M. Aquilina. Malware forensics: investigating and analyzing malicious code. Syngress, 2008.

Yazed ALsaawy is an assistant professor of Computer science at the Faculty of Computer and Information Systems, Islamic University of Madinah, Medina, KSA. He obtained his Bachelor degree in Computer Studies. Master Software Engineering and PhD degree in Computer Science from the DE Montfort University (UK). His research interests Detection Malware, Security, context-aware computing and Artificial intelligence."

Sulaiman Al amro. received his B.Sc degree in Computer Science from Qassim University, Qassim (Saudi Arabia) in 2007, M.Sc. degree in Information Technology from De Montfort University (DMU), Leicester (UK) in 2009, and Ph.D. degree in Computer Science from De Montfort University (DMU), Leicester (UK) in 2013. He is currently a working as an Assistant Professor in computer science department of Qassim University. His research interests are Network and System Security, Formal Methods and Computational Intelligence.