J Comput Virol (2010) 6:65–75
DOI 10.1007/s11416-008-0088-y

EICAR 2008 EXTENDED VERSION

# Simulating malware with MAlSim

**Rafał Leszczyna · Igor Nai Fovino · Marcelo Masera**

Received: 20 January 2008 / Revised: 26 May 2008 / Accepted: 8 June 2008 / Published online: 1 July 2008


**Abstract** This paper describes MAlSim—Mobile Agent Malware Simulator—a mobile agent framework developed to address one of the most important problems related to the simulation of attacks against information systems, i.e. the lack of adequate tools for reproducing behaviour of malicious software (malware). The framework can be deployed over the network of an arbitrary information system and it aims at simulating behaviour of each instance of malware independently. MAlSim Toolkit provides multiple classes of agents and diverse behavioural and migration/replication patterns (which, taken together, form malware templates), to be used for implementation of various types of malware (viruses, worms, malicious mobile code). The primary application of MAlSim is to support security assessments of information systems based on simulation of attacks against these systems. In this context, the framework was successfully applied to the studies on security of the information system of a power plant. The case study proved the operability, applicability and usefulness of the simulation framework and it led to very interesting conclusions on the security of the evaluated system.

## 1 Introduction

One of the approaches for security assessment of information systems is based on simulation of attacks against these systems [4]. The experiments employ the methods and tools of potential intruders and they are carried out from the position of the intruders. The approach allows to identify any potential vulnerabilities that may result from improper system configuration, known or unknown hardware or software flaws, or operational weaknesses in business processes. It leads to determination of feasibility of the attacks and their impact on the information system, on the organisation which uses it and on any other involved stakeholders [4].

Among the variety of attacks against information systems which are at disposal of intruders (and thus must be taken into account during the analyses)[1] the significant part is formed by the attacks based on *malware*, i.e. malicious software that run on a computer and make the system behaving in a way wanted by an attacker [36]. Malware attacks are the most frequent in the Internet and they pose a serious threat against information systems [34].

Malware can be categorised into the following families [36,39]:

- Viruses—programs that recursively and explicitly copy a possibly evolved version of themselves and require human interaction to propagate.
- Worms—self-replicating programs autonomously (without human interaction) spreading across a network.
- Malicious mobile code—lightweight Javascript, VBScript, Java, or ActiveX programs that are downloaded from a remote system and executed locally with minimal or no user intervention.
- Backdoors—bypassing normal security controls to give an attacker access to a computer system.

R. Leszczyna (✉) · I. Nai Fovino · M. Masera
European Commission, Joint Research Centre,
Via Enrico Fermi 2749, 21020 Ispra (VA), Italy
e-mail: rafal.leszczyna@jrc.it

I. Nai Fovino
e-mail: igor.nai@jrc.it

M. Masera
e-mail: marcelo.masera@jrc.it

[1] An approachable overview of computer attacks can be found in [1]. The updated information about system vulnerabilities is available at [34].

- Trojan horses—disguising themselves as useful programs while masking hidden malicious purpose.
- User-level RootKits—replacing or modifying executable programs used by system administrators and users.
- Kernel-level RootKits—manipulating the kernel of operating system.
- Combination malware—combining techniques of other malware families.

More detailed information on malware an interested reader can find in [39,12].

The studies on virus simulation tools span between:

- *Educational simulators*, i.e. programs demonstrating the effects of virus infection [17]. This group of programs include Virus Simulation Suite written in 1990 by Joe Hirst, which is a collection of executables, that 'simulate the visual and aural effects of some of the PC viruses' [20]. Another example is Virlab [11] from 1993, which simulates the spread of DOS computer viruses, and provides a course on virus prevention. (As it can be noticed, the programs are quite out of date, and today they would rather serve just as a historical reference.)
- *Anti-virus testing simulators*, i.e. programs which are supposed to simulate viral activity, in order to test anti-virus programs without having to use real, potentially dangerous, viruses. Unfortunately, it seams that only one solution of this type was developed [17], namely Rosenthal Virus Simulator [33]. The simulator is a set of programs which provide 'safe and sterile, controlled test suites of sample virus programs', developed for 'evaluating anti-virus security measures without harm or contamination of the system' [33]. Again the applicability of the suite is limited since it was written ten years ago.

Concerning the simulation of worms, the prevalent work was done on developing mathematical models of worm propagation [35,38,9,44], which base on epidemiological equations that describe spread of real-world diseases. The empirical approaches concentrated mainly on single-node worm spread simulators [26,25,41,31], which are dedicated to run on one machine. Only few distributed worm simulations were implemented [32,42,13]. However, in all of these approaches, also the network over which the simulated worm spreads, is simulated. Still there is a need for a simulation tool allowing simulations of malware in an arbitrary, real, physical network of computers.

Also Trojan Simulator [29] has limited applicability. It was developed for evaluating effectiveness of anti-Trojan software, and as such fulfills its purpose. However from the point of view of attack simulation, it lacks the behavioural part, since the Trojan malicious activities (e.g. stealthy task execution which consumes processor time or sending packets over network) are not simulated.

Thus it becomes evident that there are no compound frameworks for simulation of malware which would support the security assessments of information systems based on simulation of attacks.

This paper describes *MAlSim*—a new framework developed to fulfill this gap.

*MAlSim*— *Mobile Agent Malware Simulator*) is a software toolkit which aims at simulation of various malicious software in computer network of an arbitrary information system. The framework aims at reflecting the behaviours of various families of malware (worms, viruses, malicious mobile code, etc.) and various species of malware belonging to the same family (e.g. macro viruses, metamorphic and polymorphic viruses, etc.). It can simulate well-known malware (e.g. Code Red, Nimda, SQL Slammer) but it can also simulate generic behaviours (file sharing propagation, e-mail propagation) and non-existent configurations (which supports the experiments aiming at predicting the system behaviour in the face of new malware). MAlSim is a distributed simulator which simulates behaviour of each instance of malware independently. This means that if the prototype malware propagates over a network, making its copies, then the MAlSim agent dedicated to simulate this malware, also spreads across a network and creates new instances of itself.

Since the framework is based on the technology of mobile agents, the description starts with a short overview of the technology (Sect. 2). This section explains also why the paradigm of mobile agents was chosen for the development of the simulator. The next section introduces JADE (Java Agent DEvelopment Framework)—the agent platform for which MAlSim is dedicated and which provides MAlSim with mechanisms for implementing and controlling the life cycle of simulation agents. The core description of the framework starts in Sect. 4 where components of the MAlSim toolkit are explained and the notion of *malware templates* is brought in. The section describes also how experiments with MAlSim are set up. Section 5 describes malware templates in more detailed way, showing how the templates are created and used. An exemplary template of the famous virus Melissa is presented. The best way to understand how something works is to see it in action. Section 6 provides a live example of applying MAlSim for security evaluation of an information system of a power plant. Finally, Sect. 7 summarises the description of the framework.

## 2 Mobile agents

*Mobile agents* are the *software agents* able to roam network freely, to spontaneously relocate themselves from one device to another.

*Software agents* are software components, that are [2]:

- *Autonomous*—able to exercise control over their own actions.
- *Proactive* (or *goal-oriented* or *purposeful*)—goal oriented and able to accomplish goals without prompting from a user, and reacting to changes in an environment.
- *Social* (or *socially able* or *communicative*)—able to communicate both with humans and other agents.

Software agents operate on *agent platforms*. *Agent platform* is an execution environment for agents which supplies the agents with various functionalities characteristic for the agent paradigm (such as agent intercommunication, agent autonomy, yellow pages, mobility, etc.).

Agent platforms are deployed horizontally over multiple hardware devices through *containers*. On each device at least one container may be set up. Each container is an instance of a virtual machine (usually Java VM) and it forms a virtual agent network node. Containers make agent platform independent from underlying operating systems. Mobile agents are able to migrate from one container to another. Consequently, when containers are deployed on different devices, mobile agents can migrate between different devices.

Agent platforms can be imagined as agent communities where agents are managed and are given the means to interact (communicate and exchange services). Many agent communities may coexist at the same time. Depending on the implementation of the platform, agents may be able to leave one community (platform) and join another.[2]

Mobile Agent approach was chosen for the development of MAlSim because it particularly fits this purpose. Agents have much in common with malicious programs. Similarly to worms and viruses, they have the ability of relocating themselves from one computer to another. They are also autonomous as the worms are. At the same time they operate on agent platform which forms a type of sandbox facilitating their control.

## 3 JADE

MAlSim is dedicated for the JADE (Java Agent DEvelopment Framework) agent platform.

JADE is a fully Java based agent platform which complies with the FIPA[3] specifications. It is provided by means of:

- Software framework which facilitates the implementation of multi-agent systems through a middleware which supports agent execution and offers various additional features (such as a Yellow Pages service or support for agents' mobility).
- Set of graphical tools that supports the debugging and deployment phases.

JADE is licensed under Lesser General Public License (LGPL), meaning that users can unlimitedly use both binaries and code of the platform. During over seven years of its development JADE has become very popular among the members of agent community and now it is probably the most often used agent platform. JADE is continuously developed, improved and maintained, not only by the developers from the Telecom Italia Lab (Tilab), where it was originated, but also by contributing JADE community members [5,40].

Further details on the choice of JADE for the development of MAlSim can be found in [22].

## 4 MAlSim Components

*MAlSim Toolkit* provides:

- Multiple (Java) classes of MAlSim agent (extensions of JADE `Agent` class).
- Various behavioural patterns implemented as agent behaviours[4] (extensions of JADE `Behaviour` class).
- Diverse migration/replication patterns implemented as agent behaviours (extensions of JADE `Behaviour` class).

The MAlSim agent class is the basic agent code which implements the standard agent functionalities related to its management on the agent platform, its communication skills and the characteristics related to the nature of simulated malicious software. This code will be propagated across the attacked machines.

To render it operative, the code must be extended with instances of the behaviour classes and the migration/replication patterns. Depending on the chosen behaviour(s) and the migration/replication patterns, the instances of the same agent class will be created on the attacked host, or instances of another agent class from the toolkit.

The behavioural patterns comprise definitions of agent behaviours aiming at imitating malicious activities of malware (such as scanning for vulnerabilities of operating system, sending and receiving packets, verifying if certain conditions are met, etc.) but *without their harmful influence on the system*. They are implemented in Java as extensions of the `Behaviour` class provided by JADE framework. The patterns include operations such as disabling network

[2] Further information on software agents an interested reader can find in [6–8,14,15,18,21,28,43].

[3] http://www.fipa.org

[4] In agents terminology the agent's *behaviour* is a set of actions performed in order to achieve the goal. It represents a task that an agent can perform [3].

**Listing 1** Java code of `MalwareSimAgent1` class used in the template of a zero-day virus.

```
public class MalwareSimAgent1 extends MobileAgent {

  private static final long migrationDelayA = 500;
  private static final long migrationDelayB = 5000;

  String[] containerNames = {"powerplant-pc-l-100",
  "powerplant-pc-l-103", "powerplant-pc-l-104"};

  protected void setup() {
    super.setup();
    addBehaviour(new ProliferateBehaviour());
  }

  private class ProliferateBehaviour extends Behaviour {

    private int l = 0;

    public void action() {
      Random random = new Random();
      ContainerID location = new ContainerID();
      AgentController aC;
      try {
        Thread.sleep(migrationDelayA);
        if (l == 1 || l == 3)
          Thread.sleep(migrationDelayB);
      } catch (InterruptedException e1) {
        e1.printStackTrace();
      }
      try {
        location.setAddress(containerNames[l]);
        System.out.println(location.getID());
        MalwareSimAgent4 malwareSimAgent =
        new MalwareSimAgent4(location);
        aC = myAgent.getContainerController()
        .acceptNewAgent("MalSim"
          + String.valueOf(random.nextInt()),
          malwareSimAgent);
        aC.start();
      } catch (StaleProxyException e) {
        e.printStackTrace();
      }
      l++;
    }

    public boolean done() {
      return (l == containerNames.length);
    };
  }
}
```

adapter, enabling a local firewall to operate in all-block mode or starting a highly processor time consuming task, etc. They facilitate showing detrimental effects of malware activities but in contrary to their prototypes they are fully controlled. They demonstrate, for example, that after malware infection, it is no longer possible to connect to the host, or that the host's performance is affected etc. To support the demonstrative aspect of experiments also some patterns with audio-visual effects were developed. For example, to facilitate the observation of malware diffusion in the network, a sound can be played by the agent after it arrived to a new container.[5]

Migration and replication patterns describe the ways in which MAlSim agent migrates across the attacked hosts. The patterns implement malware propagation models as well as user-configured propagation schemas. The latter allow to define such characteristics as: which subnetworks of the evaluated system will be affected, in which order, at what relative time, etc.

A particular choice of one of MAlSim agent classes, extended with a chosen behavioural and migration/replication patterns is called a *malware template*, i.e. a template of malicious software. Another words, a malware template indicates a selection and configuration of Java classes (MAlSim agent, one or more behavioural patterns and one or more migration/replication patterns) selected from MAlSim Toolkit in order to simulate a particular instance of malware.

Malware template's life-cycle comprises two states:

- '*Defined*' state—when the template is described in pseudocode.
- '*Implemented*' state—when the template is actually fully implemented in Java (i.e. all the indicated MAlSim agent classes are implemented).

An exemplary malware template in the 'defined' state is presented in Sect. 5. A sample of malware template code (for a zero-day virus, see Sect. 6 for more details on the attack) is provided on Listings 1 and 2. There it can be seen that for the simulation of zero-day virus attack actually two malware agents are used: `MalwareSimAgent1` and `MalwareSimAgent2`. `MalwareSimAgent1` is a base agent, which should be launched at the 'attacker's side' JADE container. The agent creates copies of its 'children'—the instances of `MalwareSimAgent2`—providing them with the name of the target container from the list of target containers, which, in the current version of the simulation, is given explicitly. The creation of the copies is performed according to a proliferation schema defined in the `ProliferateBehaviour` class. The instances of `MalwareSimAgent2` move to the target locations and when there, they indicate they presence by playing a sound and they simulate the malicious behaviour of disrupting a driver of the network adapter. This simulation is implemented as launching the adequate system command for Linux and a Visual Basic script for Windows. In this way, it is very easy to return to the state before the experiment, by simply launching again the Visual Basic Script or running the switching-on Linux command.

[5] Interesting studies on using sound for network monitoring are described in [16].

**Listing 2** Java code of `MalwareSimAgent2` class used in the template of a zero-day virus.

```
public class MalwareSimAgent2 extends Agent {

  private Location myDestination;

  public MalwareSimAgent4(Location destination) {
    super();
    this.myDestination = destination;
  }
  protected void setup() {
    super.setup();
    addBehaviour(new Move2DestinationBehaviour(myDestination));
  }
  protected void afterMove() {
    super.afterMove();
    disableNetworkAdapter();
    InputStream in;
    try {
      in = new FileInputStream("sound.wav");
      AudioStream as = new AudioStream(in);
      AudioPlayer.player.start(as);
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
  private      class      Move2DestinationBehaviour extends
Behaviour {

    private Location myDestination;
    private int l=0;

    public Move2DestinationBehaviour(Location destination)
{
      super();
      this.myDestination = destination;
    }
    public void action() {
      myAgent.doMove(myDestination);
      l++;
    }
    public boolean done() {
      return (l==1);
    };
  }
  private void disableNetworkAdapter() {
    String os_name = System.getProperty("os.name");
    System.out.println(System.getProperty("user.dir"));

    if (os_name.toLowerCase().lastIndexOf("linux") != -1)
      try { // linux
        String line;
        String cmd = "ifdown eth0";
        Process p = java.lang.Runtime.getRuntime().exec(cmd);
        BufferedReader input = new BufferedReader(
            new InputStreamReader(p.getInputStream()));
        while ((line = input.readLine()) != null) {
          System.out.println(line);
        }
        input.close();
      } catch (Exception err) {
        err.printStackTrace();
      }
    else // windows
      try {
        String line;
        String command = "cmd /c start DisabilitaLAN.vbs";
        System.out.println(command);
        Process p = java.lang.Runtime.getRuntime().exec(command);
        BufferedReader input = new BufferedReader(
            new InputStreamReader(p.getInputStream()));
        while ((line = input.readLine()) != null) {
          System.out.println(line);
        }
        input.close();
      } catch (Exception err) {
        err.printStackTrace();
      }
  }
}
```

At the same time, the usage of mobile agents prevents from any other unpredicted consequences of the simulation, as the simulated malware is separated from the system by means of JADE environment and JADE containers.

Currently the repository of malware templates contains just several malware templates in the 'implemented' state, which are the basic malware implementations for zero-day viruses and worms. However new malware templates are planned to be implemented in a foreseeable future. At first malware templates for most interesting (from the point of view of the technique used for propagation but also regarding the payload) representatives of known malware are going to be defined (such as Yamanner, W32/Mydoom, W32/Blaster). Large enough repository of such templates will allow to extract the generic behaviours of malware (file sharing propagation, e-mail propagation, exploits) into separate malware templates.

*MAlSim setup* comprises the following phases:

1. An attack scenario is withdrawn from repository. An attack scenario is a sequence of steps taken during attack. It describes the whole 'script' of an attack, written for all participants (the attacker, the victims, the third parties). An exemplary attack scenario is provided in Sect. 6.
2. According to the chosen scenario an appropriate malware template is selected from the repository and configured. If none of existing templates fits the attack scenario, a new MAlSim template developed.
3. Creating a live instance of malware template involves extending a MAlSim agent with a migration schema (through adding agent behaviours from the repository) and a malicious behaviour.

At the current step of the development of MAlSim, the setup is done manually. In the future studies at introducing some automation to the setup process will be performed.

The experiments are controlled through the graphical interface of JADE. Using the interface, the operator can manage the whole life cycle of agents. For example he/she can launch new agents, suspend them or remove. As shown in Fig. 1 the interface provides the view at the available agent

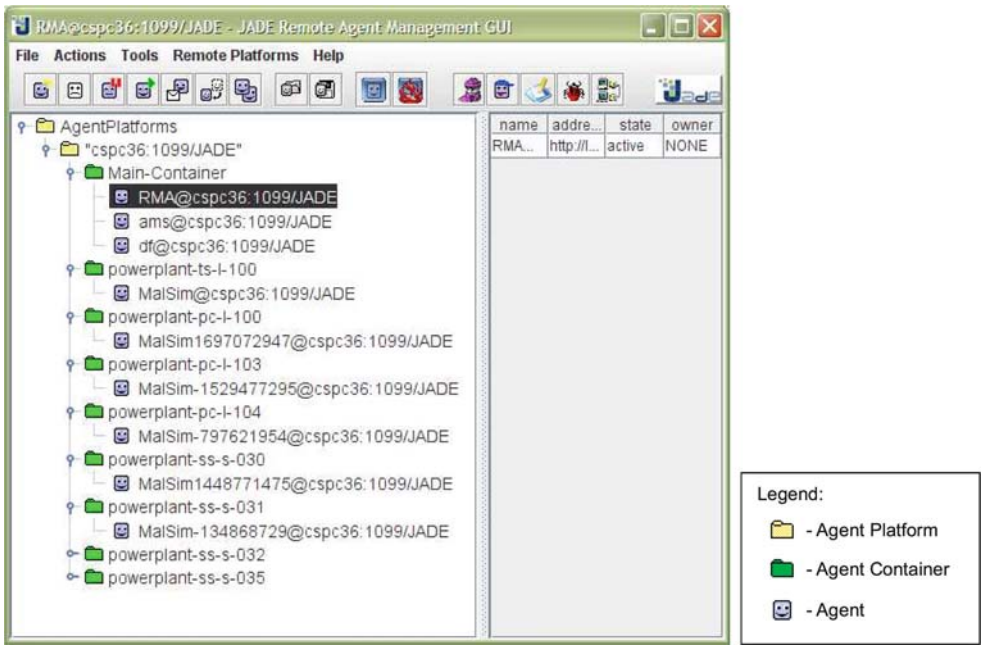


**Fig. 1** MAlSim Framework takes advantage of JADE GUI for control and observation of experiments

platforms and the containers installed on them. Each container is installed on another host participating in experiments, so from the point of view of the interface, that container represents a host. The graphical console shows which agents are present on each container. The operator can see how agents are created, they migrate, or they leave the platform. In this sense the graphical console facilitates observation of the diffusion of the simulated malware.

JADE, being a distributed agent platform supporting mobility of agents, provides MAlSim with all means for its deployment over all hosts participating in the simulation of malware. The deployment is realised through JADE containers (see Fig. 2). Java-based JADE is flexibly installable on various operating systems. During the security evaluation of a power plant (see Sect. 6) it was successfully deployed over diverse distributions of Linux (Debian, Ubuntu, CentOS) and Microsoft Windows.

More technical details of the environment can be found in [24].

As it was depicted in Sect. 1 malicious software migrate from one computer to another using network connections or portable data storage. They infect files (e.g. executables, word processing documents, etc.) or consist of lightweight programs that are downloaded from a remote system and executed locally with minimal or no user intervention (typically written in Javascript, VBScript, Java, or ActiveX). MAlSim on the other hand uses the migration mechanisms embedded in the agent platform.

In the default configuration (used for the MAlSim implementation) these mechanisms are realised over Java Remote Method Invocation protocol on port 1099. This has a negative impact on the fidelity of the simulation. Thus it is planned to develop agent behaviours aiming at minimising this difference. One solution could be for example not to allow MAlSim agent migrate until a transport channel used by the prototype malware was opened. As a result, MAlSim agent, even if 'physically' moving through the connection on 1099 port, will behave as relocating through a HTTP or POP3 connection, etc.

## 5 Malware Templates

As it was already mentioned in Sect. 4 a composition of a particular MAlSim agent class with behavioural and migration/replication patterns constitutes a malware template. The malware templates aim at reflecting the behaviours of various families of malware (worms, viruses, malicious mobile code, etc.) and various species of malware belonging to the same family (e.g. macro viruses, metamorphic and polymorphic viruses, etc.). Moreover apart of mimicking the well-known malware (such as Melissa, Code Red, Nimda, SQL Slammer), they allow simulations of generic behaviours (file sharing propagation, e-mail propagation) and their non-existent configurations. In this way a non-existent malware can be simulated, such as zero-day viruses, to more extensively evaluate the security of an information system.

During development of malware templates various information sources are used. To the most popular belong: [10,27,37].

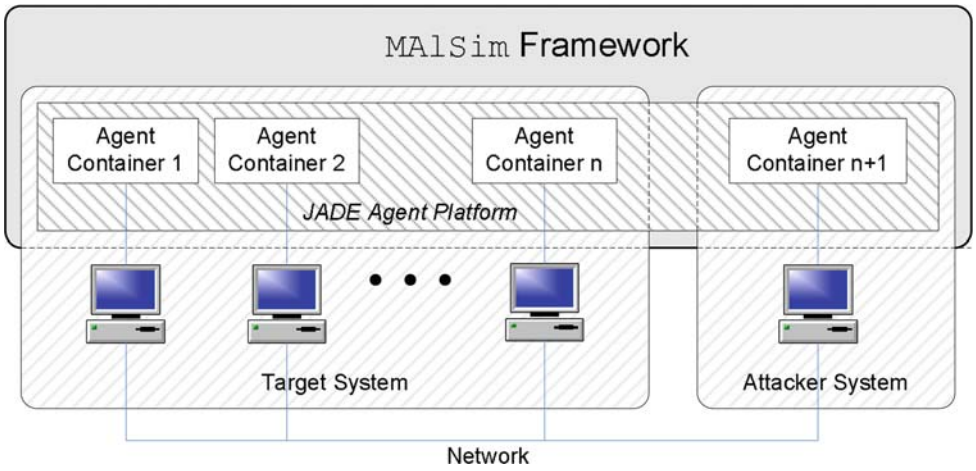


**Fig. 2** MAlSim deployment

As it can be seen on the example of the Melissa template (see below) each template defines:

- *Initial event* of the malware life cycle (a 'birth' of malware).
- *Trigger*—the overall conditions to be satisfied to allow the malware to operate.
- *Malicious actions* of the simulated malware.

These definitions drive the development of the code of MAlSim agent classes and agent behaviour classes.

Listing 3 shows the pseudocode of the malware template for simulation of the virus Melissa. The template was created based on the descriptions from [10,27,37]. The template is going to be implemented in the foreseeable future.

## 6 Case study: employing MAlSim in the security evaluation of a power plant IT system

MAlSim was applied for the experiments aiming at evaluation of the security of a power plant infrastructure.[6]

To achieve full control over the experiments and to prevent detrimental consequences which in case of critical infrastructures could have a very serious impact on many stakeholders, a secure isolated environment for attack simulations was created based on one hundred twenty hosts, the network equipment necessary to interconnect them (which includes sixteen network switches), as well as SCADA devices set up over physical hydrologic installation. In this environment, the information system of the power plant was reconstructed with very high fidelity. The identical subnetworks were created. All the key workstations of the power plant were copied in one-to-one relation. It means each of the workstations was reflected into one host of the simulation environment. Only stations of the Intranet were approximated with a lower number of hosts, but this was without loss of generality. In the reconstruction, the same network addresses were used, the same software installed, the same configurations of firewalls applied etc. More details of the environment and the reconstructions can be found in [23,24].

In this simulation environment the network setting of the power plant was reconstructed (mirrored) which comprised (Fig. 3):

- Process Network, which interconnects diverse subsystems of the energy production process.
- Field Network, which interconnects controllers and field devices.
- The corporate network (Intranet).
- Wireless LAN network.
- Demilitarised Zone (DMZ).

The JADE framework was deployed over the hosts mirroring Process Network and the Intranet. On each of the hosts a representative JADE container was installed. The experiments' control centre associated with JADE main-container, was located on the host from the Threat and Attack Simulator area of the simulation environment. From there, the simulated attacks were launched, controlled and monitored.

In this setting the simulation of a *zero-day* virus attack was performed. A zero-day (or zero-hour) attack is a computer threat that exposes undisclosed or unpatched computer application vulnerabilities. Zero-day attacks take advantage of computer security holes for which no solution is currently available. Zero-day exploits are released before the vendor

[6] An existent, fully operative combined cycle electric power plant was reconstructed and evaluated during the experiments. Unfortunately, the contractual regulations for this project require the details of the site to remain confidential.

**Listing 3** Pseudocode of the malware template for simulation of the virus Melissa.

```
Initial event:  Sending e-mail with file called LIST.DOC, which contains passwords for X-rated websites.
Trigger:  Opening the file LIST.DOC in Microsoft Word.
Action 1:  Propagating to other computers.

   1. CONNECT(MAlSim)
   2. IF "HKEY_CURRENT_USER\Software\Microsoft\Office\"→"Melissa?" EQUALS "...by Kwyjibo" THEN END
      // checking if the routine has been executed previously on the current machine
   3. OPEN(MS Outlook)
   4. MAPI_GET(userProfile)
      // getting user profile to use MS Outlook
   5. CREATE(eMailMessage)
   6. FOR {c=0; c≤50; eMailMessage.addresse = msOutlook.addressBook.contact[c]};
      // setting the message with up to 50 addresses from MS Outlook Address Book
   7. eMailMessage.subject = "Important Message From msWord.document.author"
   8. eMailMessage.body = "Here is that document you asked for ...  don't show anyone else ;-)"
   9. eMailMessage.attachments[0] = msWord.document.this
      // attaching the active WORD document to the email message
  10. SEND(eMailMessage)

Action 2:  Modifying Word documents.

   1. IF system.time.minutes EQUALS system.date.day AND (msWord.event EQUALS documentOpened) OR msWord.event EQUALS
      documentClosed) THEN msWord.document.INSERT(" Twenty-two points, plus triple-word-score, plus fifty points for
      using all my letters.  Game's over.  I'm outta here.")
      // inserting a sentence into an infected document if the number of minutes past the hour corresponds the day of
      the month (e.g.  May 3rd, 11:03) and if the document is opened or closed at the appropriate minute
   2. INFORM(MAlSim)

Action 3:  Infecting other Word documents on the user's computer.

   1. IF (msWord.event EQUALS documentCreated) msWord.newDocument.INSERT_MACRO(Melissa)
      // infecting other documents
   2. INFORM(MAlSim)

Action 4:  Hiding the activity.

   1. if msWord.version NOT EQUALS "97" THEN GO TO 6
   2. msWord.menu.DISABLE(Tools→Macro)
      // preventing listing the macro / VBA module in MS Word 97 to manually check for infection.

// setting MS Word 97 not to warn or prompt while saving the NORMAL.DOT or while opening a document with macros in
it:

   3. msWord.options.DISABLE("Prompt to save Normal template")
   4. msWord.options.DISABLE("Confirm conversion at Open")
   5. msWord.options.DISABLE("Macro virus protection")
   6. if msWord.version EQUALS "2000" THEN msWord.menu.DISABLE(Macro→Security)
      // preventing changing the security level in MS Word 2000
   7. INFORM(MAlSim)
```

patch is released to the public. A zero-day exploit is usually unknown to the public and to the product vendor.

An attack scenario was developed and based on this scenario the simulation was performed.

The scenario of the attack is as follows:

> A power plant operator working on a PC located in the power plant's Intranet browses the Internet and gets accidentally infected by a virus which has been just launched in the recent hours. This is a new type of virus, not just a slight modification of an existing one. For this reason and because of the fact that the virus is so recent, it is yet unknown to the antivirus community (zero-day virus). Its signature is not stored in any of antivirus databases.
> The virus infects programs on the user's PC and, taking advantage of the fact that unlimited traffic between the hosts in the Intranet is allowed, it infects also the remaining hosts of the Intranet. Later on the user, unconscious of the fact that his/her PC is infected by the virus, opens the VPN connection to a host in Process Control network. Now the virus has a free passageway to the critical subnetwork of the power plant network. It moves through it and starts infecting the

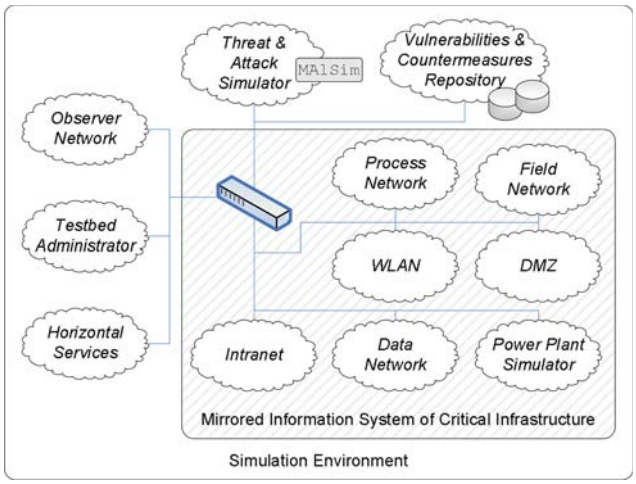


**Fig. 3** Simulation environment

computers in the Process Control network. Simultaneously, the adverse effects of the virus begin to be apparent. The computers become less effective, the applications raise errors and stop functioning, and the network connections are lost.

The general aim of this attack is to infect as many computers in the Internet as possible and to cause their malfunctioning. The attack is not particularly oriented against the power plant system, however when reaching the network of the power plant, the virus can reach the Process Control Network and Intranet subsystems.

In the simulation, the MAlSim agent had been launched at main-container and after that it was creating its copies gradually on the hosts in the Intranet and progressively in Process Network, starting from SCADA Server. After this propagation wave, the copies of MAlSim which were created at all the hosts through which it passed, were deactivating the hosts' network cards, making any network-related operation impossible.

As a result, the following services were affected:

- *Power Generation Control*—controlling and monitoring of the power production process. The viral infection and the due loss of connection with the direct controllers of the power generation devices, made impossible controlling of the power production process from Process Network. The operators were forced to use older, low level control infrastructure.
- *Power Generation Data Acquisition*—providing information necessary for the power plant supervision and for production planning. In the time between the virus outbreak and the system recovery, the data could not be collected. The operators were forced to use the alternative low level process control and monitoring infrastructure and to make production plans in non automated way. The information generated by the service is also delivered to the following cooperators, for which the interruption in the delivery of the data can become alarming:
  - High voltage power transmission and dispatching company, which transports the energy over the territory of the country.
  - End-user power distribution companies, which deliver the energy from the cross-country transmission system to the final user.
  - A government organisation which manages the electric market of the country.
- *Anomaly Diagnosis*— monitoring and analysis of vibrations of power production devices (primarily – the gas turbine), in order to predict or early detect faults or malfunctions. This service allows, for example, to predict faster utilisation of a device, allowing to make a decision of its replacement much (at least several weeks) in advance. Since the full system recovery of Process Network (based on restoring the last safe system state from backup copies) should not take more than three days (at maximum!), the loss of the anomaly diagnosis related information in the time, shall not result in any serious consequences.
- *Gas Exhaust Management*—providing information on the quality of gas emissions to the atmosphere, to the interested third parties. Provision of this service is imposed by law. Without the service, a plant cannot obtain the authorisation for energy production or the continuation of the production. Severity of the threat in regard to this service depends on the particular regulations of the country. It means, how the regulations refer to the lack of data for, at maximum, three days period (maximal system recovery time, see the previous bullet). In general restitution of the data with the estimations based on the proceeding and the following periods, and the production plan for the period of the interruption of data delivery, should suffice.
- *Remote Maintenance*—such as software patching, updating from Intranet and the Internet (!) by an authorised company. The impact of the virus in relation to the service is obvious – the software maintainers have to come to the site anyway, to remove the effects of the infection.

Summarising, the effects of this particular virus infection, though critical, were not dramatic. The power plant could continue its operation normally, from the point of view of power production process. The damages were mostly related to the interruption of data delivery, and to the necessity of performing less automated control over the production process.

This is because the payload of the simulated virus aimed only at deactivating network adapters of the infected

computers, causing 'only' the loss of connectivity. However, if another, more malicious version of the virus was developed, which, for example, would have been able to interfere with the protocol (such as MODBUS or DNP3 [19,30]) through which actual commands are sent to the Field actuators, then it could cause the anomalies in power production process.

Fortunately, the probability of the occurrence of such event is very low. To develop such a dedicated virus, an advanced level of the recognition of the power plant infrastructure (for example which protocols are used) is required, and good knowledge of SCADA protocols. Even more than these, it is difficult to develop a completely new virus, which will spread quickly enough to overpass malware detection engines.

Finally, it must be noted, that it is very difficult to prevent from the zero-day virus attack, as its strength is based on its urgency and unexpectedness. Most of antimalware software, being signature based, will be not prepared for the detection of this attack, and will let the virus spread over the networks. A possible solution for protection from this type of attacks could be to use anomaly detection based malware detection engines.

Further details about the MAlSim simulations performed in order to evaluate security of critical infrastructures can be found in [23,24].

## 7 Conclusions

The paper presented MAlSim—Mobile Agent Malware Simulator, developed to address the demand for malware simulation tools to be applied for security evaluations of information systems.

The framework is based on the technology of mobile agents, which appears to be particularly suitable for this application due to numerous similarities between agents and malicious programs (such as mobility, autonomy, etc.) and because of the features of agent platforms which facilitate performance of experiments.

MAlSim Toolkit provides multiple classes of MAlSim agent and diverse behavioural and migration/replication patterns, to be used for implementation of various malware. These components, taken together, form malware templates. An exemplar malware template for the famous virus Melissa was presented in Sect. 5.

At its current state, the MAlSim's repository of malware templates contains just basic malware implementations for zero-day viruses and worms, which were applied during the studies on computer security of a power plant. However, the repository will be successively extended with new agent classes and behaviours.

Another future task is to improve the fidelity of simulation by developing agent behaviours aiming at reducing the impact of the usage of default JADE communication mechanisms realised over Java Remote Method Invocation protocol.

The framework was successfully applied to the studies on security of a power plant [23,24], proving its operability, applicability and usefulness. The experiments showed the impact of a potential zero-day virus infection on the critical infrastructure and led to other important conclusions [23,24].

## References


1. Anderson, R.: Security Engineering: A Guide to Building Dependable Distributed Systems. Wiley, New York (2001)
2. Bellifemine, F., Caire, G., Trucco, T., Rimassa, G.: Jade—a white paper. Tilab (2003a, September)
3. Bellifemine, F., Caire, G., Trucco, T., Rimassa, G.: Jade programmers guide. Tilab (2003b, February)
4. Bishop, M.: Computer Security: Art and Science, 1st edn. Addison Wesley Professional, Reading, MA, USA (2003)
5. Caire, G.: JADE tutorial: application-defined content languages and ontologies. Tilab (2002, June)
6. Carzaniga, A., Picco, G.P., Vigna, G.: Designing distributed applications with a mobile code paradigm. In: Proceedings of the 19th International Conference on Software Engineering. Boston, MA, USA. http://citeseer.ist.psu.edu/carzaniga97designing.html (1997)
7. Chess, D., Grosof, B., Harrison, C., Levine, D., Parris, C., Tsudik, G.: Itinerant agents for mobile computing. IEEE Personal Commun. **2**(5), 34–49. http://citeseer.ist.psu.edu/article/chess95itinerant.html (1995)
8. Chess, D., Harrison, C., Kershenbaum, A.: Mobile agents: Are they a good idea? (RC 19887 (December 21, 1994 - Declassified March 16, 1995)). IBM Research, Yorktown Heights, New York. http://citeseer.ist.psu.edu/chess95mobile.html (1994)
9. Ellis, D.: Worm anatomy and model. In: Worm '03 Proceedings of the 2003 ACM workshop on rapid malcode, pp. 42–50. ACM, New York, NY, USA (2003)
10. F-Secure. F-Secure virus description database. (http://www.f-secure.com/v-descs/ (last access: January 18, 2008))
11. Faistenhammer, T., Klöck, M., Klotz, K., Krüger, T., Reinisch, P., Wagner, J.: October. Virlab 2.1. Internet. http://kklotz.de/html/virlab.html (last access: October 29, 2007)) (1993)
12. Filiol, É.: Computer Viruses: from Theory to Applications. Springer, France (2005)
13. Filiol, É., Franc, E., Gubbioli, A., Moquet, B., Roblot, G.: Combinatorial optimisation of worm propagation on an unknown network. Int. J. Comput. Sci. **2**(2), 124 – 131. http://vx.netlux.org (last access: March 7, 2008) (2007)
14. Franklin, S., Graesser, A.: Is it an agent, or just a program?: a taxonomy for autonomous agents. Intelligent agents III. agent theories, architectures and languages (ATAL'96), vol. 1193. Springer, Berlin. http://citeseer.ist.psu.edu/franklin96is.html (1996)
15. Fuggetta, A., Picco, G.P., Vigna, G.: Understanding code mobility. IEEE Trans. Software Eng. **24**(5), 342–361. http://citeseer.ist.psu.edu/fuggetta98understanding.html (1998)
16. Gilfix, M., Couch, A.L.: Peep (the network auralizer): Monitoring your network with sound. In: Lisa '00: Proceedings of the 14th USENIX Conference on System Administration, pp. 109–118. USENIX Association, Berkeley, CA, USA (2000)
17. Gordon, S.: Are good virus simulators still a bad idea? Network Security **1996**(9), 7–13 (1996)

18. Gray, R.S., Kotz, D., Cybenko, G., Rus, D.: Mobile agents: motivations and state-of-the-art systems (TR2000-365). Dartmouth College, Hanover, NH. http://citeseer.ist.psu.edu/gray00mobile.html (2000)
19. Group, D.U.: A forum for supporters of the distributed network protocol. Internet. http://www.dnp.org/ (last access: March 14, 2008) (2008, December)
20. Hirst, J.: Virus simulation suite. Internet (1990)
21. Jansen, W., Karygiannis, T.: NIST special publication 800-19-mobile agent security. http://citeseer.ist.psu.edu/jansen00nist.html (2000)
22. Leszczyna, R.: Evaluation of agent platforms Ispra, Italy: European Commission, Joint Research Centre, Institute for the Protection and security of the Citizen (2004, June)
23. Leszczyna, R., Fovino, I.N., Masera, M.: Malsim—mobile agent malware simulator. In: Proceedings of First International Conference on Simulation Tools and Techniques for Communications, Networks and Systems (SIMUTools 2008). Association for Computing Machinery (ACM) Press, New York (2008a, March)
24. Leszczyna, R., Fovino, I.N., Masera, M.: Security evaluation of IT systems underlying critical networked infrastructures. (Accepted for First International IEEE Conference on Information Technology (IT 2008), Gdansk, Poland, 18–21 May 2008) (2008b)
25. Liljenstam, M., Nicol, D.M., Berk, V.H., Gray, R.S.: Simulating realistic network worm traffic for worm warning system design and testing. In: Worm '03: Proceedings of the 2003 ACM workshop on rapid malcode, pp. 24–33 (2003)
26. Liljenstam, M., Yuan, Y., Premore, B., Nicol, D.: A mixed abstraction level simulation model of large-scale internet worm infestations. In: Mascots '02: Proceedings of the 10th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (mascots'02), p. 109. IEEE Computer Society, Washington, DC, USA (2002)
27. McAfee. McAfee virus information. Website. (http://uk.mcafee.com/virusInfo/ (last access: January 18, 2008))
28. Milojicic, D.S.: Trend wars: Mobile agent applications. IEEE Concurrency **7**(3), 80-90. http://dlib.computer.org/pd/books/pd1999/pdf/p3080.pdf (1999)
29. Mischel Internet Security. Trojan simulator. Internet. http://www.misec.net/trojansimulator/ (last access: October 29, 2007) (2003)
30. Modbus-IDA. MODBUS application protocol specification v1.1b. http://www.modbus.org/specs.php (last access: March 14, 2008) (2006)
31. Moore, D., Shannon, C., Voelker, G.M., Savage, S.: Internet quarantine: Requirements for containing self-propagating code. In: Infocom 2003. Twenty-Second Annual Joint Conference of the Ieee Computer and Communications Societies, vol. 3, pp. 1901–1910 (2003, April)
32. Perumalla, K.S., Sundaragopalan, S.: High-fidelity modeling of computer network worms. acsac 00, pp. 126–135 (2004)
33. Rosenthal Engineering. Rosenthal virus simulator. Internet (1997)
34. SecurityFocus. SecurityFocus vulnerability database. http://www.securityfocus.com/bid (last access: January 17, 2008)
35. Sharif, M.I., Riley, G.F., Lee, W.: Comparative study between analytical models and packet-level worm simulations. In: Pads '05: Proceedings of the 19th workshop on principles of advanced and distributed simulation, pp. 88–98. IEEE Computer Society, Washington, DC, USA (2005)
36. Skoudis, E., Zeltser, L.: Malware: Fighting malicious code. Prentice Hall Professional Technical Reference, Upper Saddle River, New Jersey, USA (2003)
37. Symantec. Symantec security response. (http://www.symantec.com/security_response/ (last access: January 18, 2008)
38. Symantec Research Labs 2005. Symantec worm simulator. Internet
39. Szor, P.: The art of computer virus research and defense, 1st edn. Addison Wesley Professional, Reading, MA, USA (2005)
40. Telecom Italia Lab. Java Agent DEvelopment Framework. (http://jade.tilab.com/)
41. Wagner, A., Dübendorfer, T., Plattner, B., Hiestand, R.: Experiences with worm propagation simulations. In: Worm '03: Proceedings of the 2003 ACM workshop on rapid malcode, pp. 34–41. ACM, New York, NY, USA (2003)
42. Wei, S., Mirkovic, J., Swany, M.: Distributed worm simulation with a realistic internet model. In: Pads '05: Proceedings of the 19th Workshop on Principles of Advanced and Distributed Simulation, pp. 71–79. IEEE Computer Society, Washington, DC, USA (2005)
43. Yee, B.S.: A sanctuary for mobile agents. In: Proceedings of the DARPA Workshop on Foundations for Secure Mobile Code. Monterey, USA. http://citeseer.ist.psu.edu/article/yee97sanctuary.html (last access: May 08, 2006) (1997, March)
44. Zou, C.C., Gong, W., Towsley, D.: Worm propagation modeling and analysis under dynamic quarantine defense. In: Worm '03: Proceedings of the 2003 ACM Workshop on Rapid Malcode, pp. 51–60. ACM, New York, NY, USA (2003)