Antagonistic Agents in the Internet: Computer Network Warfare Simulation

Igor Kotenko
St. Petersburg Institute for Informatics and Automation,
39, 14th Liniya, St. Petersburg, Russia
ivkote@iias.spb.su

Alexander Ulanov
St. Petersburg Institute for Informatics and Automation,
39, 14th Liniya, St. Petersburg, Russia
ulanov@iias.spb.su

***Abstract** - In the paper the approach for the simulation of counteraction between malefactors and defense systems in the Internet is considered. We try to model these antagonistic actors as software agents' teams. To simulate the teams' warfare it is proposed to use various computational models (from analytical and packet-based to virtual and testbeds). The main attention is drawn to the application of agent-oriented simulation based on the packet-based imitation of network security processes. Such approach provides acceptable fidelity and scalability in representing the attack and defense mechanisms. The approach is examined on an example of "Distributed Denial of Service" attacks and defense simulation. We consider different phases of antagonistic teams' operations – learning, decision making and counteracting, including the adaptation of one team to the actions of opposite team.*

Keywords: Network centric warfare, software and communications technology, agents, security, simulation.

1 Introduction

Today we are the witnesses of growing dependence of all sides of our vital functions from the Internet and other information technologies. The further use of these technologies becomes impossible without an appropriate solution about adequate security mechanisms.

The Internet is constantly influenced now by malefactors' attacks (viruses, worms, etc.). These attacks often succeed. So, the current state of counteraction between malefactors and defense systems can be characterized as "a game of network cats and mice" [19].

The malefactors-professionals to achieve their goals in a cyber-environment can use sophisticated strategies for the realization of different security threats. These strategies can contain a set of various actions: (1) Gathering necessary information, detection of vulnerabilities and applied defense tools; (2) Investigating methods to overcome defense; (3) Suppression, bypass or deception of defense systems; (4) Exploiting vulnerabilities and accessing resources, privilege escalation, and realizing threats; (5) Hiding the tracks of activity and creating "backdoors".

Therefore, computer network defense in present conditions needs to fulfill in real-time a complex of different measures: (1) Implementation of defense mechanisms that correspond to the defined security policy (including proactive attack prevention, malefactor disinformation, hiding and camouflage of important resources and processes, etc.); (2) Gathering and analysis of data about the state of computer system due to information processing from different sources; (3) Detection of anomaly activity, not legitimate actions, attacks and intrusions; (4) Prediction of intentions and possible malefactor actions; (5) Direct response to the intrusions, including malefactor's misleading due to false components to exposure and define his (her) goals; (6) Reflexive management of malefactor's behavior, reinforcement of critical defense mechanisms; (7) Elimination of intrusion consequences, discovered vulnerabilities and adaptation of defense system to further intrusions.

Unfortunately the present theoretical basis for information security in large-scale systems does not allow researchers to formalize adequately the mentioned set of processes. Though the researchers can represent particular defense mechanisms, the understanding of security components as a holistic system is a very difficult task. This understanding depends on many dynamical interactions between particular security processes and cyber-counteraction between different antagonistic elements. It is especially right, taking into account the present evolution of the Internet into decentralized distributed environment where a huge number of cooperating and antagonistic software agents exist and interact.

Let us examine the problem of comprehensive investigation of information security processes on an example of warfare between malefactors (realizing one of the most critical classes of computer attacks – "Distributed Denial of Service" (DDoS)) and defense mechanisms against these attacks.

To start DDoS attack a malefactor needs to compromise many hosts (zombies) to execute on them the Denial of Service software that is targeted to some victim hosts. The principal part of such attacks consists in sending to the victim a large amount of packets (UDP and ICMP flood, Smurf, Fraggle), too long packets (Ping of Death), the incorrect packets (Land), the large amount of laborious requests (TCP SYN), etc. [18].

The design and implementation of effective DDoS defense system is a very complicated problem. The effective defense includes the mechanisms of attack prevention, attack detection, tracing the attack source and attack protection. Adequate victim defense can only be achieved by the cooperation of different distributed components. So, the DDoS problem requires a distributed cooperative solution which involves a set of defense components [18].

Our goal is to suggest a common agent-based approach for the investigation and elaboration of defense methods and to produce well-grounded recommendations on the choice of defense mechanisms that are the most efficient in particular conditions. The rest of the paper is structured as follows. *Section 2* outlines the common approach for simulation. *Section 3* describes antagonistic agent teams counteracting in the Internet to realize attacks and defense. *Section 4* presents the software environment developed for simulation. *Section 5* demonstrates an example of warfare between agent teams including phases of agent learning, decision making and acting. *Conclusion* outlines main results and future work guidelines.

2 Simulation approach

We try to use the agent-oriented approach for simulation of security processes in the Internet. It supposes that the cybernetic counteraction is represented as the interaction of different teams of software agents. The aggregated system behavior becomes apparent by means of local interactions of particular agents in dynamic environment that is defined by the model of computer network.

We distinguish at least two agent teams: the team of agents-malefactors and the defense team. The agents from the same team collaborate to achieve the joint intention (to realize the threat or to defense the network).

It is assumed that competing agents gather information from different sources, operate with fuzzy (or probabilistic) knowledge, forecast the intentions and actions of opponent, estimate possible risks, try to deceive each other, and react on opponent's actions.

The choice of behavior for each team depends on the chosen goal of functioning. The choice of every step of a team behavior is defined dynamically depending on the opposite team actions and the state of environment.

Each team acts in the conditions of limited information. Every team member might have different information about actions done by other team members. Therefore, the model of agent behavior must be able to represent the incompleteness of information and the possibility of accidental factors. Besides, the agent's behavior depends on information that the team has and on its distribution on the set of particular agents of the team [3].

The models of agent's functioning are to foresee, what each agent knows, what task has to be solved and to which agent it must address its request to receive such information if it is outside of its competence. The messages of one agent are to be represented in such terms that are understandable by other agents.

The use of ontologies is the one of the most perspective approaches to structure the distributed knowledge of agents. As for every application domain the information security ontology represents the partially normalized set of notions that are to be used by other agents. Besides the relation of partial order the nodes of this structure have other relations peculiar to the application domain. The given ontology defines the subset of notions that various agents use for cooperative solving of stated tasks. Each agent uses a certain part of application domain ontology.

Each agent's specialization is represented by a subset of ontology nodes. Some of nodes can be shared by a pair or more of agents. Usually only one of these agents has the detailed description of this node. Exactly this agent is the owner of the corresponding knowledge base fragment. At the same time some part of the ontological knowledge base is shared for all agents. This part is to be the shared context (shared knowledge) for agents.

It is supposed that agents are to be able to realize the mechanisms of self-adaptation and evolution during the functioning process. The team of agents-malefactors evolves with the aid of generating new instances and types of attacks and attack scenarios to overcome the defense subsystem. The team of defense agents adapts to the malefactors' actions by changing the security policy and forming new instances of defense methods and profiles.

The strategies of agents' functioning can be represented by various formalisms, e.g. on the basis of a family of stochastic attribute grammars (and its interpretation by state machines) and hidden markov models.

The concept model of cybernetic agents' counteraction includes:

(1) Ontology of application domain containing application notions and relations;

(2) Protocols of teamwork (for the malefactors' team and the defense team);

(3) Models of individual, group and team behavior of agents;

(4) Communication component for agent message exchange;

(5) Models of environment – the computer network, including topological and functional components.

It is proposed to use a family of various models to research the processes of cybernetic counteraction.

The choice of particular models depends on the necessary fidelity and scalability of modeling and simulation. For example, analytical models let imitate the global processes happening in Internet (for example, virus epidemics). But these models describe modeled processes only on an abstract level. Packet-level simulation gives the opportunities to imitate proceeding processes with high fidelity. It allows to represent the attack and defense actions as packet exchange which precisely specifies the functioning on data link, network, transport and application layers. The greatest fidelity is archived with the hardware testbed. But it succeeds in modeling the sufficiently limited fragments of agents' interactions.

The approach realized in the paper is based on packet-level simulation using tools for network processes imitation as basic level of the simulation environment.

The following studies are used as the basis for the modeling and simulation of malefactors and defense systems counteraction in the Internet: agent-oriented simulation; agent teamwork; reasoning systems based on forecasting of opponent intentions and plans; reflexive processes; game theory; modeling and simulation of networks attacks; security processes modeling; adaptive systems and evolutionary computation.

The main basis for the research is the agent teamwork theory. There are three well-known approaches to the formalization of the agent teamwork – joint intentions theory [4], shared plans theory [10] and the hybrid approaches [12, 21] which use the combination of joint intentions and shared plans theories. A lot of teamwork approaches are implemented in various multi-agent software, e.g. GRATE*, OAA, CAST, RETSINA-MAS, COGNET/BATON, Team-Soar, etc.

Another fundamental component of the research is represented by the studies in the area of reasoning systems about opponent intentions and plans on the basis of current situation estimation [2, 14, 24, 25]. There were published the studies on determining the malefactor's plans during the intrusion detection [7, 8]. It is proposed to use the ideas of agent plans recognition on the basis of stochastic formal grammar recovery algorithms [9].

The important components in this research are the methods of reflexive processes theory [17], game theory [3] and control in conflict situations [5].

Authors used the methods of agent actions scenario specification which are based on the stochastic attributive formal grammars [9]. These methods are correlated with the colonies of cooperative distributed grammars and grammar models of multi-agent systems [15].

The teams of malefactors and defense agents are to adapt to hardware and software reconfiguration, traffic changes and new types of defense and attacks on the basis of past experience. Therefore it is important to take into account the present studies in the area of adaptation and self-learning of agents [1, 11].

The approach for teamwork proposed in the paper is based on the joint use of the elements of joint intentions theory, shared plans theory and hybrid approach.

The agent teamwork is assumed to be organized due to the shared plan of actions with the following features [16]:

(1) The group plan demands the agent team to come to agreement to fulfill the set of given instructions;

(2) Agents have to take the commitment relative not only to the individual actions but to the actions of other agents and to the actions of the whole group;

(3) The plan of group activity might include the plans of individual agents for the given action and the subgroup plans;

(4) During teamwork realization the agents have to achieve the agreement with the instructions due to communications. They also have to coordinate their intentions.

The structure of agent team is described in terms of group and individual roles hierarchy. The leaves of hierarchy correspond to the roles of individual agents, the intermediate nodes – to the group roles.

The specification of action plans hierarchy is made for every role. The following elements are defined for every plan: initial conditions, when the plan is offered for fulfillment; the conditions with which the plan stops being fulfilled; actions executed on the team level as a part of the shared plan. The joint activity is obviously expressed for the group plans.

The team members have the shared mental model. Agents can create the "snapshots" of mental state of the whole team due to joint intentions on the different abstract levels. The hierarchy of intentions is defined jointly by the team members in order to achieve the common goal. It is the consequence of agent commitments with each other.

The mechanisms of agent interaction and coordination are based on the three groups of procedures [21, 16]:

(1) providing the consistency of actions;

(2) agents' functionality monitoring and recovery;

(3) communication selectivity support (to choose the most "useful" communication acts).

3 Agent teams

3.1 Attack agents

The agents are divided into two classes: "daemon" and "master". Daemons are attack executors. Master coordinates them.

On the preliminary stage, daemons and master are deployed on available (already compromised) hosts. The important parameters are the quantity and "distribution" of agents. Then the phase of team establishing takes place. Daemons send to master the messages with information that they are alive and ready to work. Master stores the information about team members and their status. The malefactor sets the common team goal – to start the DDoS attack in the given moment of time. Master receives the attack parameters. Its goal is to send it to all available daemons. Then daemons begin to act. Their local goal is to execute the master's instruction. They begin to send attack packets to the given host in the given mode. After that it is believed that the team goal is fulfilled on this stage. Master examines daemons periodically to know that they are workable. Master controls the given attack mode by receiving the replies from daemons. When a daemon does not answer, master decides to change attack parameters. For example, it can send the commands to change the attack intensity to all or particular daemons.

Daemons can execute the attack in several modes. This influences on the possibility of defense team to detect and block the attack and to trace and defeat the attack agents. The mode is specified by the intensity of packet sending (packets per second) and the method of spoofing of sender IP address ("IP spoofing"). The method of spoofing may be as follows:

(1) Without spoofing ("no") – the real address of host (where daemon is deployed) is used;

(2) "Constant" – an address is randomly chosen, then it is used for sending the attack packets;

(3) "Random" – with every new attack packet a new address from the given range of addresses is randomly chosen. This range does not intersect with the range of addresses used in the given network;

(4) "Random real" – with every new attack packet a new address from the given range of addresses used in the network is randomly chosen.

Malefactor can stop the attack giving to master the command "stop the attack". Master resends this command to daemons, and they stop the attack.

3.2 Defense agents

In accordance with general approach there were chosen the following defense agent classes [16]: initial information processing (“sensor”); secondary information processing (“sampler”); attack detection (“detector”); filtering (“filter”); investigation (“investigator”).

In the initial moment of time the defense agents are deployed on the hosts corresponding to their roles: sensor and sampler – on the way of traffic to the defended host; detector – on any host of defended host subnet; filter – in the entrance to the defended host subnet; investigator – on any host outside of defended host subnet.

The main goal of defense team is to resist to DDoS attack. Detector watches on the goal fulfillment.

Sensor processes information of network packets and collects statistical traffic data for the defended host. Sensor calculates the amount of traffic (bits per second – BPS) and determines the addresses of hosts that make the largest traffic. Its local goal is to give that data to detector every *k* seconds.

Sampler processes the network packets and creates the model of normal functioning for the given network (in the learning mode). Then in normal mode it analyses and compares the traffic with the model of normal traffic. It picks out the addresses of hosts that do not correspond to the model and sends them to detector. The following defense methods are used in the experiments described in the paper: Hop counts Filtering (HCF), Source IP address monitoring (SIPM) and Bit per Second (BPS).

The detector local goal is to make a decision about the beginning of attack on the basis of sensor and sampler data. For example, if the BPS parameter for any address exceeds the given threshold, it is believed that attack happens. Detector sends the list of addresses to filter and investigator. That are the addresses from sensor that have BPS more than the given maximum and all addresses received from sampler.

Filter local goal is to filter the traffic on the basis of detector data.

Investigator goal is to trace and defeat the attack agents. After receiving the message from detector it examines the received addresses for the presence of attack agents and tries to defeat them. When detector decides that attack is stopped (on the basis of sensor and sampler data), it is believed that the team goal is fulfilled on this stage.

The methods used by sampler are as follows.

Hop counts Filtering (HCF) [13]. It is used the assumption that the packets from the same subnet pass through the same hops on the way from sender to receiver. The count of hops is estimated due to the packet TTL field. It is decremented on each router. The initial value of it can be 30, 32, 60, 64, 128 or 255. The special table is created in the learning mode. The table is formed on basis of requests to the defended host. It consists of IP addresses grouped by their hop count. The system calculates the hop count of incoming packet and compares it with the given value in the normal mode. If the count of hops differs that the packet is dropped.

Source IP address monitoring (SIPM) [23]. The assumption is used that in the beginning of attack there are a lot of packets which are sent from new IP addresses and directed to the defended host addresses. There is created the table of legitimate addresses in the learning mode based on clients’ requests. Both in normal and learning modes the system calculates the amount of new IP addresses for the given interval *dt* with the given shift *tshift*. This means that the amount is calculated every *tshift* seconds for the previous *dt* seconds. In the learning mode the maximum value (threshold) of new addresses amount is estimated. Then, in normal mode, if the amount of new addresses stays lower than the threshold, these addresses are stored. If the amount exceeds the threshold during several intervals (this type of aggregation is called cumulative sum method, CUSUM), then packets from new addresses are dropped.

Bit per Second (BPS). It is used the assumption that traffic from one IP address should not exceed some critical threshold. In the learning mode it is calculated the amount of transmitted bits per second (BPS) during the given interval for every client requesting defended host. The greatest BPS value (threshold) is determined. In the normal mode if the BPS parameter for some address exceeds the determined threshold then packets from this host are dropped. This parameter is calculated every *tshift* seconds for previous *dt* seconds.

The main parameters for sampler are threshold values as well as *tshift* and *dt* for SIPM and BPS. The key parameter for SIPM is also the maximum amount of intervals during which the threshold was exceeded.

4 Simulation environment

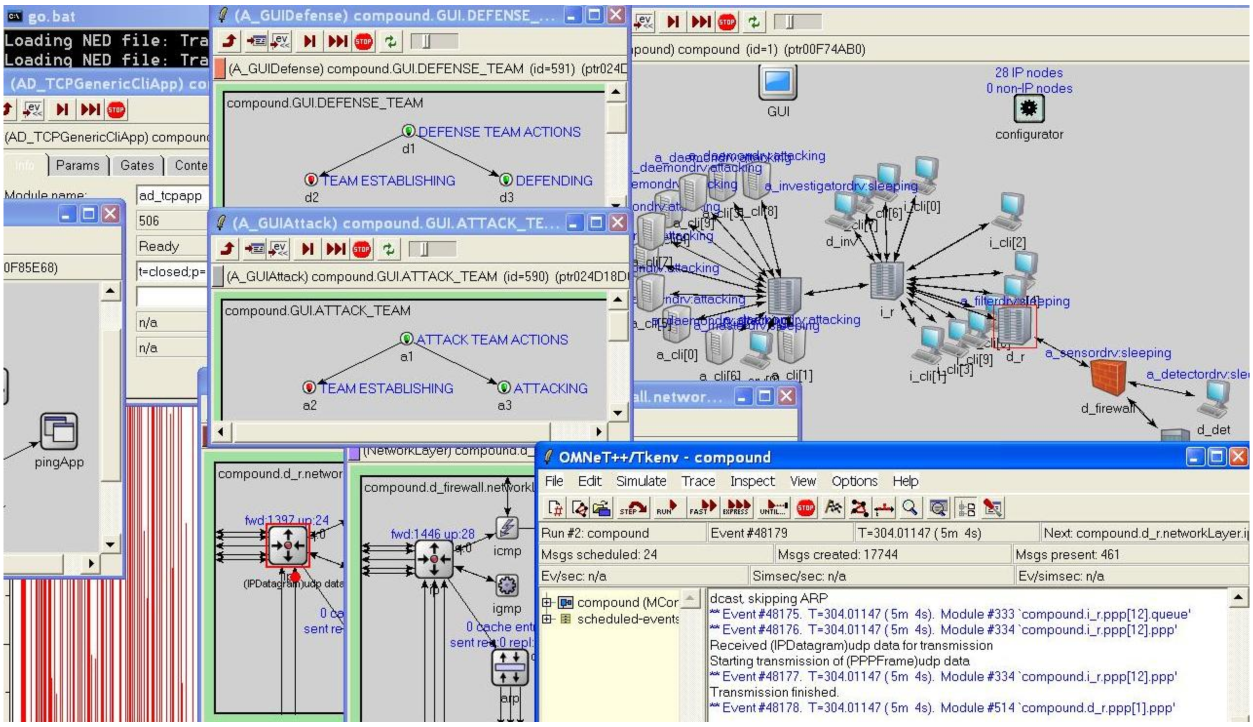
We developed our simulation environment using OMNET++ INET Framework [22].

The example of multi-window user interface of the simulation environment is depicted in Figure 1. At the basic window of visualization (Figure 1, at upper right), a simulated computer network is displayed.

The window for simulation management (Figure 1, at bottom right) allows looking through and changing simulation parameters. It is important that you can see the events which are very valuable for understanding attack and defense mechanisms on the time scale. The time scale is depicted above windows with the events description.

Corresponding status windows (on top of Figure 1, in the middle) show the current status of agent teams. It is possible to open different windows which characterize functioning (the statistical data) of particular hosts, protocols and agents, for example, at the bottom left of Figure 1, the window of one of the hosts is displayed.

At the basic window of visualization (Figure 2), a simulated computer network is displayed. The network represents a set of hosts and channels. Hosts can fulfill different functionality depending on their parameters or a set of internal modules. The routers are labeled with the sign “”. Attack agents are deployed on the hosts marked with red color. Defense agents are located on the hosts marked with green. Above the colored hosts there are the strings that indicate the corresponding state of deployed agents. The other hosts are the standard hosts that generate the generic network traffic.



```
Loading NED file: Tra
Loading NED file: Tra
```

```
Run #2: compound    Event #48179    T=304.01147 ( 5m  4s)    Next: compound.d_r.networkLayer.i
Msgs scheduled: 24    Msgs created: 17744    Msgs present: 461
Ev/sec: n/a    Simsec/sec: n/a    Ev/simsec: n/a
```

```
dcast, skipping ARP
** Event #48175.  T=304.01147 ( 5m  4s).  Module #333 `compound.i_r.ppp[12].queue'
** Event #48176.  T=304.01147 ( 5m  4s).  Module #334 `compound.i_r.ppp[12].ppp'
Received (IPDatagram)udp data for transmission
Starting transmission of (PPPFrame)udp data
** Event #48177.  T=304.01147 ( 5m  4s).  Module #334 `compound.i_r.ppp[12].ppp'
Transmission finished.
** Event #48178.  T=304.01147 ( 5m  4s).  Module #514 `compound.d_r.ppp[1].ppp'
```

Figure 1 : common representation of the simulation environment

The hosts are connected with the channels. Their parameters can be changed. They are as follows: "delay" – delay of packets propagation; "datarate" – the speed of packets transmission.

During agent design and implementation there were used the elements of abstract FIPA architecture [6]. The main idea of such representation is to provide the interaction of agents and the ability of agents' reuse. Such system description gives the possibility to see the correlations between the main elements of agent-based system.

There were used the following elements of the abstract architecture for the agents in the developed system: communication language, transport layer, agent directory. The implementation of interaction language is needed for all agents to let them transmit the messages. Agent directory is needed for the agents "master" and "detector" that coordinate the activity of agents in theirs teams. Daemon needs implementation of two transport components (for communications and attacks). The agent of filtering needs the implementation of network layer to let it apply filtering rules. Agents "sensor" and "sampler" are to have the network layer also to let them process and collect the data, for example, to create the model of normal traffic.

Agents are deployed on the hosts in the simulation environment. Their installation is fulfilled by connecting to the modules serving the transport and network layers of protocol stack simulated in OMNeT++ INET Framework.

Each network for simulation consists of three sub-networks: (1) the subnet of defense where the defense team is deployed; (2) the intermediate subnet where the standard hosts are deployed. They produce the generic (normal) traffic in the network including the traffic to defended host; (3) the subnet of attack where the attack team is deployed.

The subnet of defense (Figure 2, the hosts highlighted with green) consists of five hosts. The following agents are deployed on the first four hosts: detector, sampler, filter and investigator. The web-server which is under defense is deployed on the fifth host. The agents and the web-server are the applications installed on the corresponding hosts. The IP addresses are being installed automatically. It is necessary to fix a set of other application parameters.

Web-server is deployed on the host d_srv. The interaction port and the answer delay must be set. Detector is deployed on the host d_det. The following parameters are used for detector: the defended host IP address, the port for team interaction, the interval for sensor inquiry, and the maximum allowed data-rate to server (BPS, bit per second). Sampler is deployed on the host d_firewall (on the entrance to the server subnet). Filter is installed on the host d_r (router). Investigator is deployed on the host d_inv. For each of the last three agents, the private port, the IP address of detector and the port for team interaction must be set.

The intermediate subnet (Figure 2, not highlighted hosts) consists of N hosts i_cli[…] with generic clients. They are connected by the router i_r. The number of hosts N is the simulation parameter which can be set. The following parameters of clients must be specified: IP-address and port of server, the time of work start, the quantity and size of requests while connecting to server, the size of reply and the time of reply preparation, the idle interval.

The subnet of attack (Figure 2, hosts highlighted with red) consists of M hosts i_cli[…] with daemons deployed and one host with master deployed. The number of hosts M must be set. Master has the following parameters: port for team interaction, IP-address and port of attack target, the time of start of attack and its rate (measured in packets per second). Daemon has the following parameters: the port, masters' IP-address and port for team interaction.

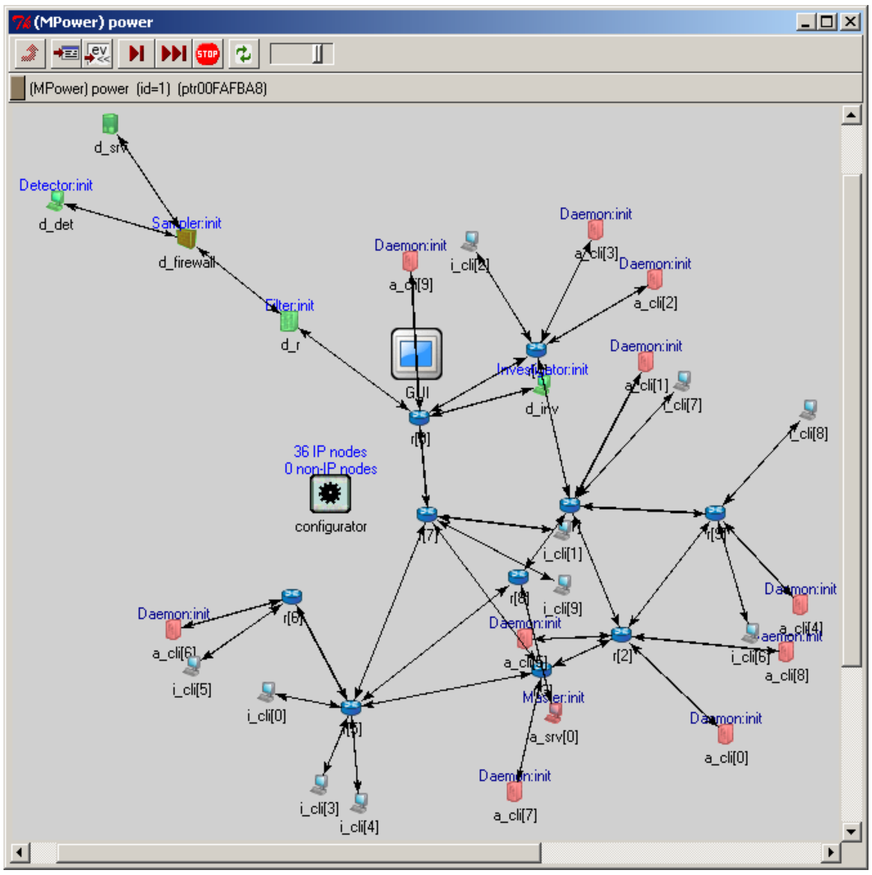
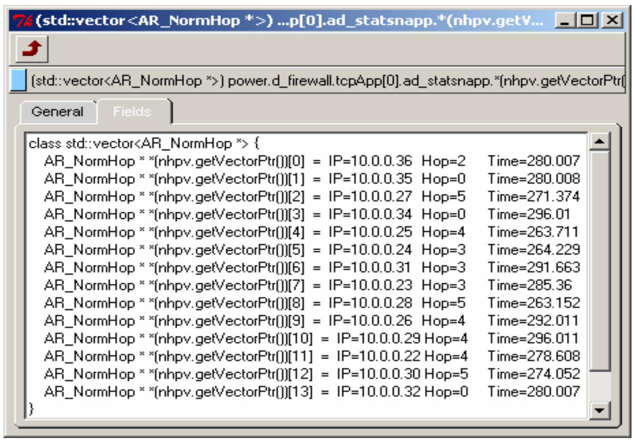


Figure 2 : example of computer network for simulation

5 Example of warfare simulation

5.1 Learning mode of operation

The main task of learning mode is to create the model of generic traffic for the given network. The clients send the requests to the server and it replies. At this time sampler analyses requests and uses them to form the models and parameters for the SIPM, HCF and BPS methods. During the learning it is possible to watch the change of traffic models for each of discussed methods.



```
(std::vector<AR_NormHop *>) ...p[0].ad_statsnapp.*(nhpv.getV...
(std::vector<AR_NormHop *>) power.d_firewall.tcpApp[0].ad_statsnapp.*(nhpv.getVectorPtr(
General  Fields
class std::vector<AR_NormHop *> {
   AR_NormHop * *(nhpv.getVectorPtr())[0]  =  IP=10.0.0.36  Hop=2     Time=280.007
   AR_NormHop * *(nhpv.getVectorPtr())[1]  =  IP=10.0.0.35  Hop=0     Time=280.008
   AR_NormHop * *(nhpv.getVectorPtr())[2]  =  IP=10.0.0.27  Hop=5     Time=271.374
   AR_NormHop * *(nhpv.getVectorPtr())[3]  =  IP=10.0.0.34  Hop=0     Time=296.01
   AR_NormHop * *(nhpv.getVectorPtr())[4]  =  IP=10.0.0.25  Hop=4     Time=263.711
   AR_NormHop * *(nhpv.getVectorPtr())[5]  =  IP=10.0.0.24  Hop=3     Time=264.229
   AR_NormHop * *(nhpv.getVectorPtr())[6]  =  IP=10.0.0.31  Hop=3     Time=291.663
   AR_NormHop * *(nhpv.getVectorPtr())[7]  =  IP=10.0.0.23  Hop=3     Time=285.36
   AR_NormHop * *(nhpv.getVectorPtr())[8]  =  IP=10.0.0.28  Hop=5     Time=263.152
   AR_NormHop * *(nhpv.getVectorPtr())[9]  =  IP=10.0.0.26  Hop=4     Time=292.011
   AR_NormHop * *(nhpv.getVectorPtr())[10]  =  IP=10.0.0.29 Hop=4     Time=296.011
   AR_NormHop * *(nhpv.getVectorPtr())[11]  =  IP=10.0.0.22 Hop=4     Time=278.608
   AR_NormHop * *(nhpv.getVectorPtr())[12]  =  IP=10.0.0.30 Hop=5     Time=274.052
   AR_NormHop * *(nhpv.getVectorPtr())[13]  =  IP=10.0.0.32 Hop=0     Time=280.007
}
```

Figure 3 : list of hosts that sent requests to server and hops to them after 300 seconds of learning

Figure 3 represents the list of hosts that sent requests to server and hops to them after 300 seconds of learning and the time of last request. As mentioned above the hop count is calculated on the basis of the TTL packet field.

Figure 4 depicts the change of new addresses amount for sampler during first 300 seconds of learning. One can see that in the beginning when clients requested server at the first time there were many new addresses (the maximum is 6 addresses, the time interval is 10 seconds, and the shift is 3 seconds). The last unknown address appeared in the region of 100 first seconds. At least, when all clients requested the server there were no new addresses.

Figure 5 represents the list of clients requested the server and considered as legitimate after first 300 seconds of learning. One can see here that in the interval between 0 and 50 seconds there were many new addresses. Figure 6 represents the change of maximum BPS (for interval 10 seconds and shift 3 seconds) after 300 seconds from the beginning of learning.

The maximum value was 1742.4 bit/s and was recorded in the area of 100 seconds. One can see also the values of BPS for clients that requested server in the current time interval. Figure 7 depicts the values of transmitted bits for every client that requested server in the interval of 10 seconds.

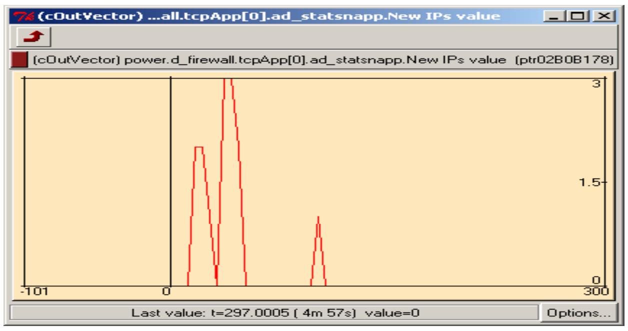
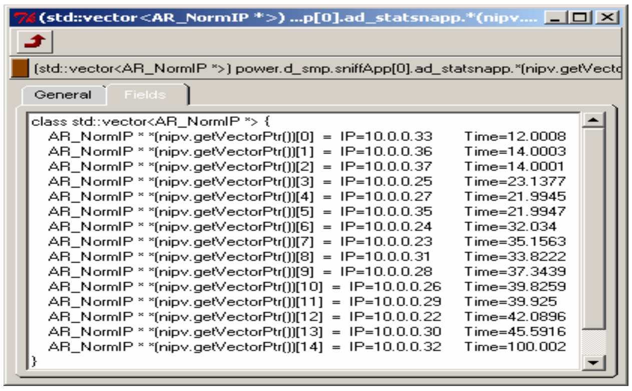


Figure 4 : change of new IP addresses amount



```
(std::vector<AR_NormIP *>) ...p[0].ad_statsnapp.*(nipv....
(std::vector<AR_NormIP *>) power.d_smp.sniffApp[0].ad_statsnapp.*(nipv.getVecto
General  Fields
class std::vector<AR_NormIP *> {
   AR_NormIP * *(nipv.getVectorPtr())[0]  =  IP=10.0.0.33       Time=12.0008
   AR_NormIP * *(nipv.getVectorPtr())[1]  =  IP=10.0.0.36       Time=14.0003
   AR_NormIP * *(nipv.getVectorPtr())[2]  =  IP=10.0.0.37       Time=14.0001
   AR_NormIP * *(nipv.getVectorPtr())[3]  =  IP=10.0.0.25       Time=23.1377
   AR_NormIP * *(nipv.getVectorPtr())[4]  =  IP=10.0.0.27       Time=21.9945
   AR_NormIP * *(nipv.getVectorPtr())[5]  =  IP=10.0.0.35       Time=21.9947
   AR_NormIP * *(nipv.getVectorPtr())[6]  =  IP=10.0.0.24       Time=32.034
   AR_NormIP * *(nipv.getVectorPtr())[7]  =  IP=10.0.0.23       Time=35.1563
   AR_NormIP * *(nipv.getVectorPtr())[8]  =  IP=10.0.0.31       Time=33.8222
   AR_NormIP * *(nipv.getVectorPtr())[9]  =  IP=10.0.0.28       Time=37.3439
   AR_NormIP * *(nipv.getVectorPtr())[10]  =  IP=10.0.0.26     Time=39.8259
   AR_NormIP * *(nipv.getVectorPtr())[11]  =  IP=10.0.0.29     Time=39.925
   AR_NormIP * *(nipv.getVectorPtr())[12]  =  IP=10.0.0.22     Time=42.0896
   AR_NormIP * *(nipv.getVectorPtr())[13]  =  IP=10.0.0.30     Time=45.5916
   AR_NormIP * *(nipv.getVectorPtr())[14]  =  IP=10.0.0.32     Time=100.002
}
```

Figure 5 : list of clients requested server and considered as legitimate after 300 seconds of learning

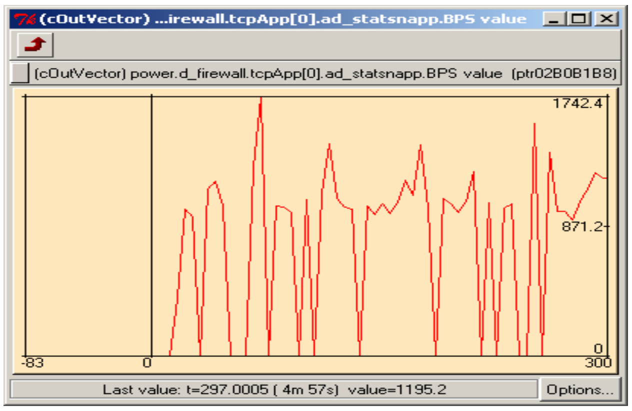
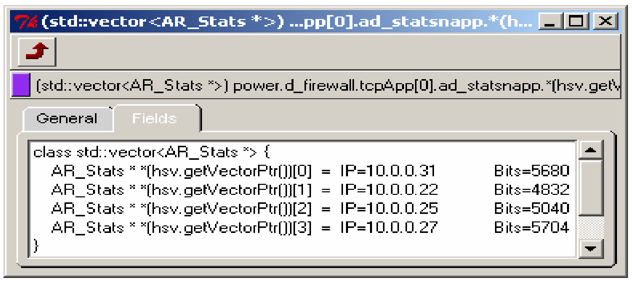

(cOutVector) ...irewall.tcpApp[0].ad_statsnapp.BPS value
(cOutVector) power.d_firewall.tcpApp[0].ad_statsnapp.BPS value (ptr02B0B1B8)
1742.4
871.2
0
-83
0
300
Last value: t=297.0005 (4m 57s) value=1195.2
Options...

Figure 6 : change of BPS parameter



```
(std::vector<AR_Stats *>) ...pp[0].ad_statsnapp.*(h...
(std::vector<AR_Stats *>) power.d_firewall.tcpApp[0].ad_statsnapp.*(hsv.getV
General   Fields
class std::vector<AR_Stats *> {
   AR_Stats * *(hsv.getVectorPtr())[0]  =  IP=10.0.0.31        Bits=5680
   AR_Stats * *(hsv.getVectorPtr())[1]  =  IP=10.0.0.22        Bits=4832
   AR_Stats * *(hsv.getVectorPtr())[2]  =  IP=10.0.0.25        Bits=5040
   AR_Stats * *(hsv.getVectorPtr())[3]  =  IP=10.0.0.27        Bits=5704
}
```

Figure 7 : values of transmitted bits

5.2 Decision making and acting

Figure 2 represents the structure of the network for simulation and the allocation of attack and defense agents. Simulation scenario is realized on the same configuration as was used for learning. The only difference is that the attack team is engaged now.
Attack team initial parameters are as follows: target_ip="d_srv" (target of attack is server d_srv); target_port="2001" (target port); t_ddos=300 (time of attack start); attack_rate=5 (intensity of attack in packets per second); ip_spoofing="no" (no IP spoofing is used).
After simulation start the clients begin to send requests to the server and it replies. This is the way the generation of generic network traffic takes place (Figure 8, interval 0 – 300 seconds).
The formation of defense team occurs after some time from start. Investigator, sampler and filter connect to detector and send it the messages that they are alive and ready to work. Detector stores this information. The attack team is formed in the same way. Daemons connect to master and report their status.
After establishing the defense team begins to function. Sampler collects traffic data and compares it with the model data that was acquired during learning mode. The addresses that are the source of anomalies are sent to detector every *n* seconds (in this scenario *n*=60). Detector makes the decision about the attack and sends to filter and investigator the addresses of suspicious hosts.
Figure 8 represents the graphs of channel throughput on the entrance to the defended network before (red) and after (blue) filter.
After 300 seconds from simulation start the attack team begins attack actions. Master examines all daemons that it knows. Then it sends the attack command to all workable daemons. This command includes address and port of attack target, intensity (distributed among daemons) and the method of IP spoofing. In this case they are: target – d_srv, port – 2001, intensity of attack for every daemon (calculated as intensity divided by the number of daemons) 5/10=0.5, spoofing – “no” (no IP spoofing). When daemons receive the command they begin to send the attack packets (Figure 8, timestamp 300 seconds).
After a while, sampler determines the suspicious hosts with the use of BPS method. The BPS parameter of these hosts exceeds normal. Detector receives the addresses of these hosts from sampler and sends them to filter and investigator. Filter sets the filtering rules and the packets from the given hosts begin being dropped (Figure 8, timestamps 400 – 600 seconds, blue graph).
Investigator tries to inspect the given hosts and to defeat the attack agents deployed there. It succeeds in defeating of 4 daemons. The string “defeated” appears above the defeated agent in the window of network structure. However the other daemons continue the attack (Figure 8, after 400 seconds, red graph).
Master examines daemons next time 600 seconds after modeling starts. It does not succeed to connect with all daemons since some of them were defeated by investigator.
Master makes the decision to redistribute the intensity of attack to keep the overall intensity on the given level. Also it decides to change the method of IP spoofing to complicate the detection and defeating of attack agents by defense team. Master sends to alive daemons the command: target – d_srv, target port – 2001, intensity – 5/(10–4)=0.83, IP spoofing method – “random”. When daemons receive the command they continue to send the

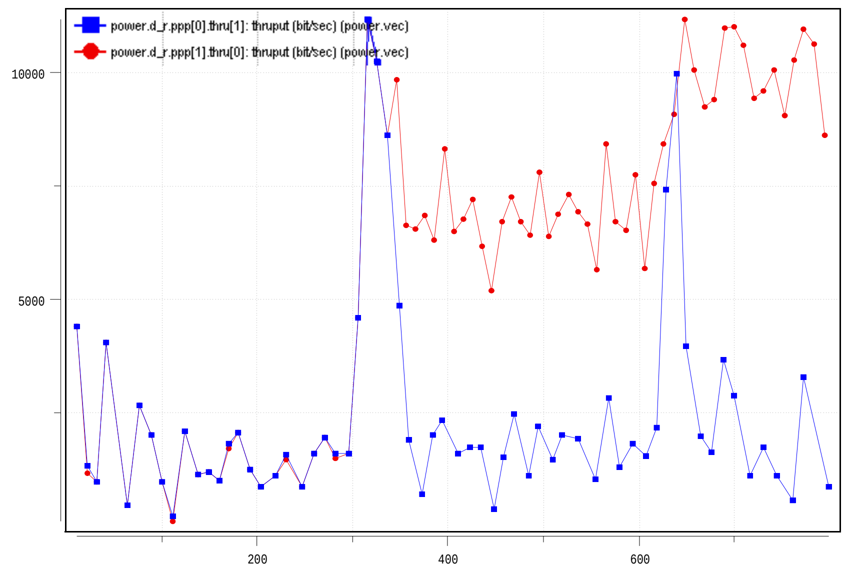


Figure 8 : graphs of channel throughput on the entrance to the defended network before (red) and after (blue) filter (bits/s to seconds)

attack packets having applied the new parameters (Figure 8, timestamp 600 seconds).
Detector sees that the input channel throughput has noticeably lowered since the traffic from attack team has raised (Figure 8, after 600 seconds). Detector does not receive the anomaly report from sampler though. This is because the method BPS used by sampler does not work fine when attacker changes the sender address in every packet. That is the reason that detector fails to confront some address with the big traffic.
Therefore detector decides to apply another DDoS defense method – SIPM. Then the large amount of new IP addresses for sampler leads to attack detection and dropping the malicious packets. This method however does not allow tracing the source of attack and investigator fails to defeat attack agents. But the attack packets are filtered and the traffic in the subnet of defended host returns to normal state.

6 Conclusions

The main results of the work we described in the paper consist in developing basic ideas on agent-based modeling and simulation of defense mechanisms against attacks and implementing the corresponding software environment.
The environment developed is written in C++ and OMNeT++. It allows to imitate a wide spectrum of real life DDoS attacks and defense mechanisms.
Different experiments with this environment have been fulfilled. These experiments include the investigation of attack scenarios and protection mechanisms for the networks with different structures and security policies. One of the scenarios was demonstrated in the paper.
Future work is connected with building more realistic environment, and conducting experiments to both evaluate computer network security and analyze the efficiency and effectiveness of security policy against different attacks.
This research is being supported by grant of Russian Foundation of Basic Research (№ 04-01-00167), grant of the Department for Informational Technologies and Computation Systems of the Russian Academy of Sciences (contract №3.2/03) and partly funded by the EC as part of the POSITIF project (contract IST-2002-002314).

References

[1] T.Back, D.B.Fogel, and Z.Michalewicz, *Evolutionary computation. Vol. 1. Basic algorithms and operators*, Institute of Physics Publishing, 2000.
[2] E.Charniak, and R.P.Goldman, A Bayesian Model of Plan recognition. *Artificial Intelligence*, V.64, N 1, 1993.
[3] A.G.Chhartishvili, Game theory modeling of information control in active systems, *Human factor in control systems*. Moscow, 2005 (in Russian).
[4] P.Cohen, H.J.Levesque, Teamwork, *Nous*, 35, 1991.
[5] V.V.Druzhinin, D.S.Kontorov, M.D.Kontorov, *Introduction into conflict theory*. Moscow, Radio i svyas', 1989 (in Russian).
[6] *FIPA*. http://www.fipa.org
[7] C.W.Geib, and R.P.Goldman, Plan recognition in intrusion detection systems, *DARPA Information Survivability Conference and Exposition*, DARPA and the IEEE Computer Society, 2001.
[8] R.P.Goldman, C.W.Geib, and C.A.Miller, A New Model of Plan Recognition, *Proceedings of the 1999 Conference on Uncertainty in Artificial Intelligence*, 1999.
[9] V.Gorodetski, and I.Kotenko Attacks against Computer Network: Formal Grammar-based Framework and Simulation Tool, *Recent Advances in Intrusion Detection. Fifth International Symposium. RAID 2002.* Zurich, Switzerland. Lecture Notes in Computer Science, V.2516, 2002.
[10] B.Grosz, and S.Kraus, Collaborative Plans for Complex Group Actions, *Artificial Intelligence*, Vol.86, 1996.
[11] D.Gu, and E.Yang, Multiagent Reinforcement Learning for Multi-Robot Systems: A Survey, *Technical Report of the Department of Computer Science, University of Essex, CSM-404*, 2004.
[12] N.R.Jennings, Controlling cooperative problem solving in industrial multi-agent systems using joint intentions, *Artificial Intelligence*, Vol.75, No.2, 1995.
[13] C.Jin, H.Wang, and K.G.Shin, Hop-count filtering: An effective defense against spoofed DDoS traffic, *Proceedings of the 10th ACM Conference on Computer and Communications Security*, 2003.
[14] H.Kautz, and J.F.Allen, Generalized plan recognition, *Proceedings of the Fifth National Conference on Artificial Intelligence*, 1986.
[15] J.Kelemen, Colonies: grammars of reactive systems, *Proceedings of AICRS'97*. World Scientific, Singapore, 1997.
[16] I.Kotenko, and A.Ulanov, Multiagent modeling and simulation of agents' competition for network resources availability, *Second International Workshop on Safety and Security in Multiagent Systems*. Utrecht, The Netherlands. 2005.
[17] V.A.Lefevre, *Reflexion*. Moscow, "Kognito-Center", 2003 (in Russian).
[18] J.Mirkovic, S.Dietrich, D.Dittrich, and P.Reiher, *Internet Denial of Service: Attack and Defense Mechanisms*. Prentice Hall PTR, 2004.
[19] *Nomad Mobile Research Centre*. http://www.nmrc.org
[20] A.A.Stogniy, and A.I.Kondrat'ev *Game theory information modeling in decision making systems*. Kiev: Naukova dumka, 1986 (in Russian).
[21] M.Tambe, Towards flexible teamwork, *Journal of AI Research*, Vol.7, 1997.
[22] *OMNeT++ homepage*. http://www.omnetpp.org/
[23] T.Peng, C.Leckie, and R.Kotagiri, Proactively Detecting DDoS Attack Using Source IP Address Monitoring, *Networking 2004*, Athens, Greece, May, 2004.
[24] M.Vilain, Getting Serious about Parsing Plans: A Grammatical Analysis of Plan Recognition, *Proceedings of the Eighth National Conference on Artificial Intelligence*, Cambridge, MA, 1990.
[25] M.P.Wellman, and D.V.Pynadath, *Plan Recognition under Uncertainty,* Unpublished web page, 1997.