Proceedings of the 9th Colloquium for Information Systems Security Education
Georgia Institute of Technology
Atlanta, Georgia, 6-9, June 2005

Modeling, Simulation, and Visualization of a Network Using the Easel Programming Language

Vojislav Stojkovic, Grace C. Steele

Abstract – Modeling, simulation, and visualization play a significant role in the study of Information Assurance and Infrastructure Security. Research in this area is generally multidisciplinary in nature and collaboratively conducted by researchers with expertise in computer science, engineering, business, mathematics, and statistics. The paper presents modeling, simulation, and visualization of a hypothetical network system using the Easel agent-oriented programming language. The network system is represented as a multi-agent system. This multi-agent approach provides a more complete understanding of network attack and defense postures, network dynamics, and the computation of network security. The paper provides a very useful and hands-on approach to teaching network security through the use of modeling. When the use of real-world systems is not practical, simulated networks can be used as educational tools. This approach provides students the freedom to experiment with network security scenarios in a safe and effective environment rather than with actual physical networks. The main contribution of the paper is that it provides an original way to calculate the level of security of a network and graphically represent it.

I. Modeling, Simulation, and Visualization

Models are adequate descriptions of reality. A model is the manifestation of reality in a controlled environment whereby the impact of changes can be determined without actually altering reality. It is very costly, dangerous and often impossible to conduct experiments using real systems. Experimenting with models can be cost effective, efficient, and safe. It is for precisely these reasons that models are used extensively in science, research, industry, commerce, and the military.

A simulation is the execution of a model, represented by a program that gives information about the system being investigated.

There are two approaches to analyzing a model:

- analytical approach and
- simulation approach.

While the analytical approach of model analysis is theoretical in nature, the simulation approach is more practical. The simulation approach is more reliable than the analytical approach and provides more flexibility and convenience. The activities of the model consist of events that are activated at specific points in time and, in this way, affect the overall state of the system. Events can occur at random time intervals making external input to the system unnecessary. Events occur autonomously and are discrete therefore there is an absence of activities between the execution of two distinct events.

Most simulations proceed with known, assumed, or hypothesized information. The biggest challenge is to simulate unobserved, ill defined or virtually invisible processes.

Visualization is the graphical representation of a data collection, often in an interactive form. Presentation of data in graphical form allows users to perceive inherent patterns and to discover relationships illustrated by these patterns. The visualization of complex relationships among a set of related data items may be applicable to a number of different situations.

In the past computational visualizations have resulted in unique scientific discoveries that are visually accessible to a wider audience than ever before. Visual expressions of knowledge can result in new levels of understanding of highly theoretical aspects of the sciences and new insights into nature itself.

The use of visualization techniques could provide many advantages to theoretical work in terms of researching, exploring, and representing both the known and the not-yet known via methods that will reveal new and unique nodes of knowledge.

Visualization tools facilitate the examination of data at a very micro level and therefore, can provide users with a better understanding of the data and serve to improve learning. Users are allowed to manipulate the data so that they can examine it from a more basic level in order to compare aspects of the data so that its meaning becomes more comprehensible.

Visualization tools facilitate the presentation of data as maps, charts, and graphs and provide users an opportunity to manipulate the data by applying sorting, subdividing, or a combination of routines to particular chunks of data.

Morgan State University
School of Computer, Mathematical, and Natural Sciences
Baltimore, MD 21251

ISBN 1-933510-99-4/05/$15.00 © 2005 IRI/CISSE

II. The Programming Language EASEL

Easel – An Emergent Algorithm Simulation Environment and Language - is a new general-purpose, modeling, simulation, and visualization programming tool. [2].

Easel can be used for describing, modeling, simulating, and visualizing systems/processes with large numbers of interaction components and/or participants that lack central control, and contain incomplete and imprecise information. Easel is intended as a research tool for handling unbounded systems, emergent algorithms and survivability architectures. Easel was designed to work with large numbers of locally interacting actors – autonomous participants of a system/process. It focuses on modeling the emergent properties resulting from the complex interactions between these actors.

Easel is a tool for use by individuals or organizations that are interested in modeling, simulating, and visualizing complex, and dynamically changing networks of components. These systems are characterized by local neighbor interactions, limited visibility of knowledge between hidden components/processes and/or larger surrounding systems/processes, and system-wide emergent properties. Easel helps to predict behavior and interactions of components/processes/systems.

Easel is currently a discrete event simulation programming language with limited support for continuous variables. It supports multiple levels of abstraction, multiple simultaneous belief systems, distributed specifications, and dynamic graphic depictions.

III. Actors

An actor is an independent process. While an actor exists, the actor:

- has its own memory, and
- requires (some) CPU time:
  - to update its internal state and
  - to interact with other actors.

Each actor may have global and local properties. The global properties of actor type are defined in the actor parameter list. The local properties of actor type are defined in the actor body. In the Easel programming language an actor is a predefined type that has the property “being threaded”.

IV. Network

We use the agent-oriented programming language Easel for modeling, simulation, and visualization of a hypothetical network. We gave the name Network to the implemented program. The Network program computes the level of risk and provides a graphic representation of the risk levels using a specific color scheme. [1], [3], [4].

The network consists of:

- nodes and
- communications between these nodes.

There are four types of nodes:

- attackers
- defenders
- uncompromised hosts, and
- compromised hosts.

The nodes are colored-coded using the following scheme:

- attackers	in	red
- defenders	in	blue
- uncompromised hosts	in	green
- compromised hosts	in	black

Communications between nodes occurs in the following manner:

- an attacker

can change an uncompromised host into a compromised host by sending a virus i.e., a green node changes into a black node;

- a defender

can change a compromised host back into an uncompromised host by sending an antivirus i.e., a black node changes into a green node;

The user has the ability to input:

- number of attackers
- number of defenders
- number of uncompromised hosts and
- number of compromised hosts

which can be defined or changed by the user. The rate of infection can also be defined - changed through user input.

Easel appears to have the capacity to simulate a network correctly and provide the user with appropriate feedback. The use of Easel to teach network simulation is a sound idea. However, users must be careful that they do not become too mired in the technical aspects of the language that they lose their focus on the actual simulated network and the information it provides.

ISBN 1-933510-99-4/05/$15.00 © 2005 IRI/CISSE

V. Implementation Details

The network is represented as a multi-agent system. The Network program is implemented as a multi-agent system in the Easel programming language. [5], [8].

The Easel program consists of:

- a sequence of agents – actors and
- a sequence of auxiliary functions

One auxiliary function is the main function.

The Easel programs may have three basic conceptual components:

- a model component to specify abstract descriptions of the entity types
- a depiction component to graphically display the results
- a simulation component to initiate and execute the simulation

The Network program has four actors:

- host actor
- arc actor
- bar actor and
- advisory actor.

A. Host Actor

The host actor:

- paints circle
- depicts circle

The global properties of host actor are: identifier, kind, and activity.

The local properties of host actor are: x and y coordinates, color, and image.

```
role: type is enum
        (attacker, defender, uncompromised, compromised);

Circle(r: number, c: pattern): drawing is
            return paint(circle(0.0, 0.0, r), c);

host(id: int, kind: role, activity: number): actor type is

 x:: number := random(uniform, 50.0, 600.0);
 y:: number := random(uniform, 50.0, 550.0);

 host_color:: pattern := role_color kind;
 host_image:: drawing := Circle(20.0, host_color);
 depict
 (
  sim.v,
  var       offset_by
            (
                        var host_image,
                        var x,
                        var y
            )
 );

 r:: number := 0.0;

 # Life cycle of a host

 for every true do
   for h: each sim.hosts do
    r :=  random(uniform, 0.0, 100.0);
    if activity > r
    then
     if kind = attacker & h.kind = uncompromised
     then
      push(sim.arcs, new(sim, arc(self, h)));
      h.kind := compromised;
     if kind = defender & h.kind = compromised
     then
      push(sim.arcs, new(sim, arc(self, h)));
      h.kind := uncompromised;
     host_color := role_color kind;
     wait 1.0;
    wait 1.0;
```

Each host is depicted – place on the screen by the depict-function.

var ensures that the depiction of the host actor is continuously updated in the display.

offset_by function ensures that the depiction of the host actor will be appropriately placed in the display.

5 attacker hosts may be depicted – placed on the screen in the following way:

```
num_uncompromised:: number := 10;

j:: int := 0;
attacker_activity:: int := 5.0;
for each 1 .. num_attackers do
 push
 (
  NetworkSim.hosts,
  new(NetworkSim, host(j, attacker, attacker_activity))
 );
 j := j +1 ;
```

ISBN 1-933510-99-4/05/$15.00 © 2005 IRI/CISSE

The network represented at the Figure 1 has:

- 5 attacker hosts
- 5 defender hosts
- 10 compromised hosts and
- 20 uncompromised hosts

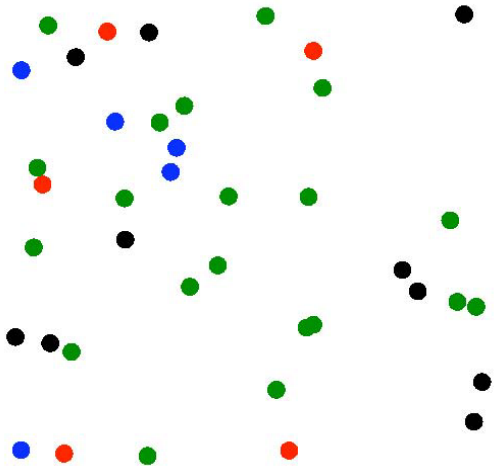

Figure 1. Network: Hosts

B. Arc Actor

Arc actor connects source host with destination host.

The global properties of the arc actor are:

- source host and
- destination host.

The local properties of the arc actor are: width, coordinates, color, and image.

```
Polyline
(
 w: number,
 x0: number, y0: number, xf: number, yf: number,
 c: pattern
): drawing is
        return paint (polyline(w, x0, y0, xf, yf), c);

arc(source: host, destination: host): actor type is

 arc_width:: number := 1.0;

 x0:: number := source.x;
 y0:: number := source.y;

 xf:: number := destination.x;
 yf:: number := destination.y;

 arc_color:: pattern := (black);
 arc_image:: drawing :=
        Polyline(arc_width, x0, y0, xf, yf, var arc_color);

 depict
 (
  sim.v,
  var arc_image
 );

 i:: int := 0;

 for every i < 257 do
  arc_color := rgb(i, i, i);
  i := i + 10;

 wait 1.0;
```

The network represented at the Figure 2. has:

- 40 hosts and
- some number of arcs.

The number of arcs depends of the network activities.

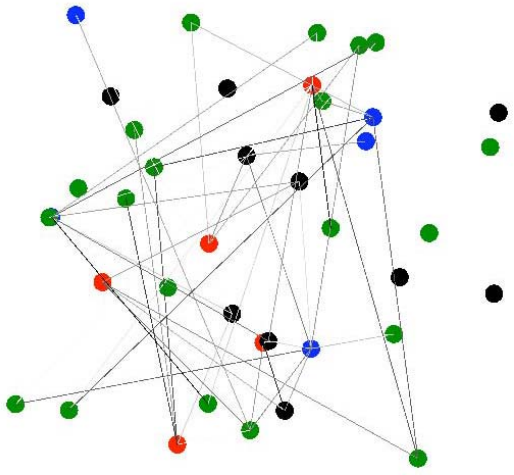

Figure 2. Network: Hosts and Arcs

C. Bar Actor

Bar actor depicts the bar image.

Bar image is a rectangle, defined by:

- x and y coordinates of the main node
- width and height of the side, and
- color.

The height is defined by:

ISBN 1-933510-99-4/05/$15.00 © 2005 IRI/CISSE

```
 - the constant factor and
 - the variable number.
The number is the number of hosts.

Polygon
(
        x: number, y: number,
        w: number, f: number, num: number,
        c: pattern
): drawing is
        return
        (
            var paint
                (
                        polygon
                        (
                                x, y,
                                x+w, y,
                                x+w, y-f*num,
                                x, y-f*num
                        ),
                        c
                )
        );

bar(x: number, y: number, which: role): actor type is

 w:: number := 30.0;
 f:: number := 5.0;
 num:: number := 0.0;

 bar_color:: pattern := role_color which;

 bar_image:: drawing :=
  Polygon
  (
   x, y,
   w, f,  (var num),
   bar_color
  );

 depict
 (
  sim.v,
  var bar_image
 );

# Life cycle of the bar

 for every true do

  num := 0.0;

  for h: each sim.hosts do

   if h.kind = which
   then
    num := num + 1.0;
   wait 1.0;
```

Four bars, where each bar represents the total numbers of:

- compromised hosts
- uncompromised hosts
- attackers and
- defenders

may be depicted – placed on the screen in the following way:

```
x:: number := 100;
y:: number := 630;
d:: number := 40;

null new(NetworkSim, bar(x, y, compromised));
null new(NetworkSim, bar(x+d, y, uncompromised));
null new(NetworkSim, bar(x+2*d, y, attacker));
null new(NetworkSim, bar(x+3*d, y, defender));
```

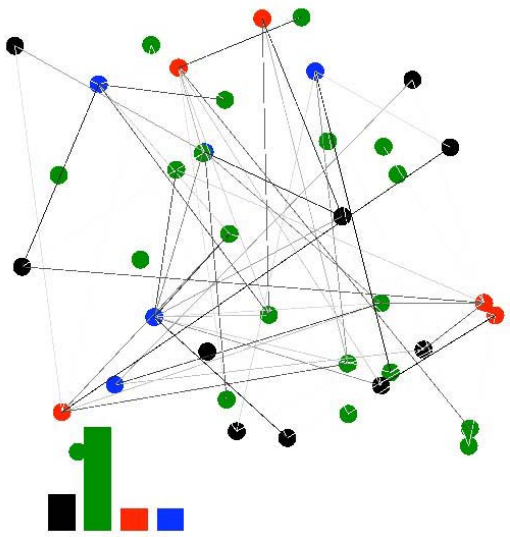

Figure 3. Network and Bars

D. Advisory Actor

Advisory actor depicts the advisory image as a user-defined, constant position in the window.

Advisory image is a circle, defined by:

- a user defined – constant radius and
- a variable advisory color.

The advisory color is computed by the risk_color function.

ISBN 1-933510-99-4/05/$15.00 © 2005 IRI/CISSE

The argument of the risk_color function is computed by the risk_calculation function.

The arguments of the risk_calculation function:
- number of compromised hosts and
- number of uncompromised hosts

are computed by advisory actor in the life cycle.

```
advisory(x: number, y: number): actor type is

advisory_color:: pattern := risk_color low;

num_C:: number := 0.0;
num_U:: number := 0.0;
num_A:: number := 0.0;
num_D:: number := 0.0;
kind:: risk;

advisory_image:: drawing :=
  Circle(50.0, var advisory_color);

depict
 (
  sim.v,
  var       offset_by
            (
             var advisory_image,
             x,
             y
            )
 );

# Life cycle of the advisory

 for every true do

  num_C := 0.0;
  num_U := 0.0;
  num_A := 0.0;
  num_D := 0.0;

 for h: each sim.hosts do

  if h.kind = compromised
  then
   num_C := num_C + 1.0;
  else
   if h.kind = uncompromised
   then
    num_U := num_U + 1.0;
   else
    if h.kind = attacker
    then
            num_A := num_A + 1.0;
    else
            if h.kind = defender
            then
             num_D := num_D + 1.0;

   kind := risk_calculation(num_C, num_U);

   advisory_color := risk_color kind;

 wait 1.0;
```

The advisory image, represented by the different colors indicates the current level of security in the network:

- red	is	severe
- orange	is	high
- yellow	is	elevated
- blue	is	guarded
- green	is	low

The advisory image may be depicted – placed on the screen in the following way:

```
x:: number := 400;
y:: number := 600;

null new(NetworkSim, advisory(x, y));
```

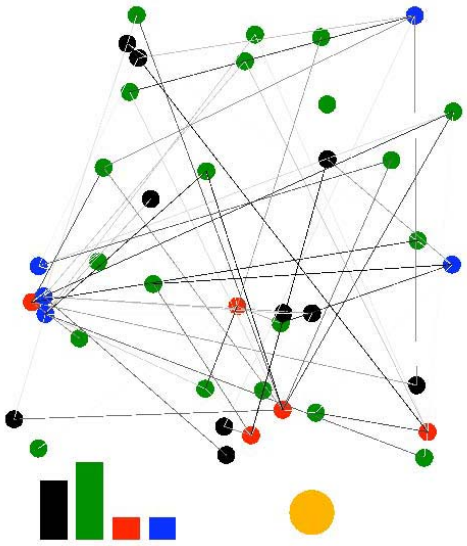

Figure 4. Network and Advisory

E. Main – Net Function

Simulation component initiates and executes the simulation.

Network is declared as the simulation type.
Network simulation type has the following three user-defined attributes:

ISBN 1-933510-99-4/05/$15.00 © 2005 IRI/CISSE

- v - view, used to portray the cells
- hosts - list of hosts, used to reference hosts and
- arcs – list of arcs, used to reference arcs.

```
Network: simulation type is
        v       :: view := ?;
        hosts   :: list := new list host;
        arcs    :: list := new list arc;
```

Simulation is initiated and starts execution by:

```
num_attackers: int is 5;
num_defenders: int is 5;
init_num_uncompromised: int is 20;
init_num_compromised: int is 10;

Net
(
        num_attackers,
        num_defenders,
        init_num_uncompromised,
        init_num_compromised
);
```

The Net procedure is the facilitator that manages the simulation.

The Net function:

- creates a simulation
- creates the simulation view
- activates all actors and
- waits for the simulation to complete

```
Net
(
        num_attackers: int,
        num_defenders: int,
        num_uncompromised: int,
        num_compromised: int
): action is

NetworkSim:: Network := new Network;

NetworkTitle:: string := "Network";
NetworkBackgroundColor:: pattern := white;
NetworkObject:: drawing := nil;

NetworkSim.v := new view
 (
        NetworkSim,
        NetworkTitle,
        NetworkBackgroundColor,
        NetworkObject
 );

 null make_window(NetworkSim.v, 0);

# Work-around: loop control variables

 j:: int := 0;
 # Create the attackers

 for each 1 .. num_attackers do
  push
  (
        NetworkSim.hosts,
        new(NetworkSim, host(j, attacker, 1.0))
  );
  j := j + 1;

# Create the defenders

 for each 1 .. num_defenders do
  push
  (
        NetworkSim.hosts,
        new(NetworkSim, host(j, defender, 5.0))
  );
 j := j + 1;

# Create the uncompromised

for each 1 .. num_uncompromised do
  push
  (
   NetworkSim.hosts,
   new(NetworkSim, host(j, uncompromised, 5.0))
  );
 j := j + 1;

# Create the compromised

for each 1 .. num_compromised do
  push
  (
   NetworkSim.hosts,
   new(NetworkSim, host(j, compromised, 1.0))
  );
  j := j + 1;

# Create the bars

  null
   new(NetworkSim, bar(100.0, 630.0, compromised));
  null
   new(NetworkSim, bar(140.0, 630.0, uncompromised));
  null
   new(NetworkSim, bar(180.0, 630.0, attacker));
  null
   new(NetworkSim, bar(220.0, 630.0, defender));

# Create the advisory
null new(NetworkSim, advisory(400.0, 600.0));
```

ISBN 1-933510-99-4/05/$15.00 © 2005 IRI/CISSE

```
# Wait for children to terminate

 wait NetworkSim;
```

F. Auxiliary Procedures and/or Functions

As in any other program there are some auxiliary procedures and/or functions.

- role_color function computes for a given kind, the appropriate color.

```
role_color(kind: role): pattern is

 if kind = attacker
 then
  return (red);
 else if kind = defender
       then
        return (blue);
       else if kind = uncompromised
             then
               return (green);
             else if kind = compromised
                   then return (black);
```

- risk_color function computes for the given kind of risk, the appropriate color.

```
risk_color(kind: risk): pattern is

 if kind = severe
 then return (red);
 else if kind = high
       then return (orange);
       else if kind = elevated
             then return (yellow);
             else if kind = guarded
                   then return (blue);
                   else if kind = low
                         then return (green);
```

- risk_calculation function computes for the given number of compromised and uncompromised hosts the appropriate risk level.

```
risk_calculation
 (num_C: number, num_U: number): pattern is
 if num_C > 0.5 * (num_C + num_U)
 then
  return (severe);
 else
   if num_C > 0.4 * (num_C + num_U)
    then
     return (high);
    else
      if num_C > 0.3 * (num_C + num_U)
       then return (elevated);
       else if num_C > 0.2 * (num_C + num_U)
             then return (guarded);
             else return (low);
```

VI. Conclusion

This study is important for a number of reasons. First, organizations are constantly changing and change brings many new risks. Simulation, modeling, and visualization tools are useful for minimizing risk and managing change in the environment. Second, organizations use data and information to make important decisions. Simulation, modeling, and visualization tools can be employed to support decision-making and prediction.

The Network program is quite interesting because:
- it demonstrates the concept of emergent property
- uncompromised/compromised types of hosts are randomly changed
- arcs are generated randomly;
- bars are changed
- advisory color is changed
- individual actors do not have initial knowledge that each is only a part of the whole collection of actors– the multi actor – agent system. [8].

This Easel system implementation of the Network is unique in that it provides a simple way to model a network and render a dynamic picture of the simulation for observation purpose. The Network program is useful for both teaching and further research.

VII. References

[1] A.M. Christie, “Network Survivability Analysis Using Easel”, *Technical Report*, CMU/SEI, Pittsburgh, PA, December 2002, pp. 1-56.
[2] A.M. Christie, D. Durkee, D.A. Fisher, and D.A. Mundie, “Easel Language Reference Manual and Author’s Guide”, *Technical Report*, CMU/SEI, Pittsburgh, PA, February 24, 2003, pp. 1-199.
[3] A.M. Christie, and D.A. Fisher, “Simulating the emergent behavior of complex software-intensive organizations”, CMU/SEI, Pittsburgh, PA, pp. 1-9.
[4] M. DeSantis, “Using Easel to Study Complex Systems”, *news@sei interactive*, CMU/SEI, Pittsburgh, PA, 3Q, 2001, pp. 1-3.
[5] D.A. Fisher, “Design and Implementation of EASEL – A Language for Simulating Highly Distributed Systems”, CMU/SEI, Pittsburgh, PA, pp. 1-11.
[6] W. Lupton and V. Stojkovic, “Solving Incomplete and Incorrect Information Problems Using Conditional Planning, Execution Monitoring, and Situated Planning Agents”, *The Proceedings of 12th Annual ASEET Symposium*, Monterey, CA, 27-30 July, 1998.

ISBN 1-933510-99-4/05/$15.00 © 2005 IRI/CISSE

[7] M.D. Roblyer and J. Edwards, *Integrating Educational Technology into Teaching (*2nd edition), Merrill - Prentice Hall, Upper Saddle River, NJ, 2000.
[8] V. Stojkovic and W. Lupton, "Software Agents – A Contribution to Agents Specification", ISECON 2000, Philadelphia, PA, Nov 09-12, 2000.

ISBN 1-933510-99-4/05/$15.00