Can Network Security be Fun? An agent-based Simulation Model and Game proposal

Frode Petter Gilberg

Master's Thesis
Master of Science in Information Security
30 ECTS
Department of Computer Science and Media Technology
Gjøvik University College, 2006

Institutt for
informatikk og medieteknikk
Høgskolen i Gjøvik
Postboks 191
2802 Gjøvik

Department of Computer Science
and Media Technology
Gjøvik University College
Box 191
N-2802 Gjøvik
Norway

Abstract

Too often, people's knowledge about network and information security is built from their own experiences with accidents, or through "learning by burning". This could be a very costly affair.

Most Internet users lack the motivation and knowledge needed to have the appropriate protection. It is hard to seek knowledge from the literature when you do not know what to look for.

Using design research, a network security game based on a complex agent-based simulation was designed, and we wanted to see if it was possible to build a game prototype within a time-period of five months. We also wanted to investigated if this game could be fun to play.

A game design, both model and prototype, are presented and developed.

Our study showed that it is possible to build a game prototype and that it can be built in five months if the developer have the prior technical skills required, and that (s)he are willing to use more than 1200 man-hours to complete the game.

The degree of enjoyment experienced in the current version, does not seem to be high enough for people to keep playing the game for a longer period of time. The game should be extended with more challenges and better graphics, music and sound effects. In addition, further studies should focus on competitive elements, like multi-player and real-time attack/defend scenarios, to make it more enjoyable.

Further studies must be conducted with more participants to tell with enough degree of confidence that a network security game is fun to play.

Sammendrag

Altfor ofte lærer man om nettverk- og informasjonssikkerhet gjennom egne, dårlige, erfaringer, eller gjennom såkalt “learning by burning”. Dette kan være en kostbar affære.

De fleste internettbrukere mangler motivasjonen og kunnskapen som skal til for å beskytte deres systemer godt nok, og det kan være vanskelig å finne kunnskapen i litteraturen når man ikke vet hva man skal søke etter.

Ved å bruke design research metoden designet vi et nettverkssikkerhetsspill basert på en kompleks agent-basert simulasjon. Vi ønsket å se om det var mulig å lage en slik spill-prototype innenfor en tidsperiode på fem måneder. Vi ønsket også å undersøke om dette spillet kunne oppleves som morsomt å spille.

Et spill design, både modellen og en prototype blir presentert og utviklet.

Vår undersøkelse viser at det er mulig å utvikle en spillprototype og at dette kan gjøres innenfor den nevnte tidsrammen hvis utvikleren har de tekniske forkunnskaper som skal til og er villig til å bruke mer enn 1200 mannetimer på oppgaven.

Graden av erfart underholdning i den nåværende versjonen ser ut til å være for lav til at noen ønsker å spille spillet over en lengre periode. Spillet burde utvides med flere utfordringer og bedre grafikk, lyd og lydeffekter. I tillegg bør man i videre studier fokusere på andre konkurranseelementer som for eksempel en flerspillerversjon med både angresps- og forsvars- scenarioer for å gjøre spillet mer morsomt.

Videre studier må utføres med flere deltakere for at man men nok grad av sikkerhet kan si at et nettverksspill er morsomt å spille.

Preface

First of all I would thank Prof. Einar Snekkenes at the Norwegian Information Security laboratory (NISLab) for supervising the planning and work of this study, and for pointing me in the right directions at different phases throughout this project.

I would also like to thank Espen Torseth at The Norwegian Centre for Information Security (NorSIS) who helped me with technical issues related to network management and network security.

Finally, I would like to thank the players who participated in the experiment and who gave valuable feedback to this project.

Your help and guidance were highly appreciated.

Frode Gilberg, 2006/06/25

Contents

List of Figures

List of Tables

1 Introduction

1.1 Revisions

Date	Comment
9.dec 2005	Title changed from “Evaluating Security Decisions using Simulations” to “Using Games to Improve Network Security Decisions”
11.dec 2005	Project proposal ready for evaluation
31.jan 2006	First chapter outline proposal
21.may 2006	Outlining of all chapters ready for evaluation
23.may 2006	Title changed to ’Can Network Security be Fun?’

Table 1: Document history.

1.2 Definitions

Word/Concept	Explanation
IS	Information Security
IA	Information Assurance
IS Domain	The boundary inside which IS decisions are made
ISP	Internet Service Provider
IT	Information Technology
OS	Operating System

Table 2: Definitions.

1.3 Topic covered by this thesis

The purpose of this study is to look at existing information- and network- security games, network simulators and popular commercial simulation games to identify game elements that fit well into a network security context.

A game proposal will be discussed and what it should include to be valid for network security gaming and how elements can be transformed into an enjoyable game. A simulation model will be designed and a game prototype will built from scratch.

It will be investigated to what extent this game can be built within a five months period and if it will be, or has the potential to be, fun to play.

Keywords: network security, computer game, agents, simulation game, model, enjoyment.

1.4 Problem description

> “Mrs. Experience is a tough teacher. She gives the test first and the lesson afterward”
> -Unknown

Too often, knowledge about network- and information- security is accumulated through first-hand experiences, or “learning by burning”. The implications of security breaches

can be enormous in both cost and reputation, and in some worst-case scenarios, companies or enterprises might never recover from these events.

Numerous security breaches could have been avoided if people had better knowledge so that they could make better security decisions, both prior to an incident, and afterward to reduce the damage caused by these incidents.

Most users know little about information security. Large scale attacks are often accomplished through the use of hosts in home networks, which in turn were compromised because its owners did not know how to secure their systems.

Although there are tools, games and simulators, that can be used to increase people's knowledge, they seem to focus on special areas of interests, and are often aimed at experts in these areas. Also, access to these tools is often restricted.

Knowledge can be built through literature, but it can be hard to know what to look for, when you don't know what the problem is, or what the challenges are. Average users seldom look for literature to improve their knowledge, due to lack of motivation or awareness.

1.5 Justification, motivation and benefits

Internet citizens need a tool that could motivate them to seek information about network- and information- security and how to improve the security of their own, and other's, infrastructure.

Information security is not only the responsibility of those who maintain corporate or Internet networks. Large scale attacks on the Internet often origins from hosts in private- and home- networks, and it is therefore everyone's responsibility to take information security seriously, even though they don't think their assets are of any value to others.

With information technologies, processes interact in complex ways, and it can be hard to figure out the implications of security related decisions. Sometimes it's even hard to recognize that a decision is really security related at all, as events, security related or not, can be interconnected in very complex patterns. What is really a small event can easily escalate into a major disaster.

A tool is needed to inform people about these patterns. A better understanding of event correlation and the actions needed to control events would improve decision making, and hence, reduce the impact of information security violations, on people, businesses and the society as a whole.

1.6 Stakeholders

The stakeholders for this project should be everyone involved in accessing public IT networks, from the average user to companies, Internet Service Providers (ISPs) and managements concerned with network security and vulnerabilities. In short, those who want to increase the general awareness and knowledge, and hence reduce the number of events that violates the integrity, confidentiality or availability of IT networks.

Educational institutions would also benefit from a tool that helps them improve student's knowledge in this area.

1.7 Research questions

Working with network security involves dealing with a wide range of tasks and configurations. A game need to include many of these elements. Building a computer game

is a large undertaking and involves many processes, from the creative activity of building a conceptual idea to the design and development process involving technological knowledge in multiple disciplines. Creating a game is one thing, making it fun to play is something else. The threshold for success and failure is a fine one. It must be investigate if the game based on a new complex simulation model could be fun to play, and/or identify what would make the game (more) enjoyable. Questions for this study are;

- **Q1**: Is it possible to build a network security game within a short time limit?
- **Q2**: Will it be fun to play?

1.7.1 Is it possible to build a game within a short time limit?

The total time period of this study was six months and it was conducted by one participant; the author of this document. The idea is to build a game based on a complex state structure of networks and network units. A complex model should make it possible to re-create attacks seen in real life systems and hence increase the game realism. Related issues is how trivial the game can be to be considered valid for network security gaming and if this game can be built within the time limits of this study.

1.7.2 Will it be fun to play?

In game development, to attract players, a big concern is how to make a game fun to play. In this study, this would involve identifying tasks in the network domain that introduces exciting challenges and game play. How tasks can be handled in the game's user interface must be figured out, and considerations must be made in all layers of the software architecture, the data layer, logic layer and graphical front end, to meet user expectations. The level of enjoyment experienced by players should indicate how "finished" the game is considered to be at the end of this study.

2 Summary of claimed contributions

In this study we describe the process of building a network security game from scratch, and show that this can be done within a time period of six months.

Achieving enjoyment in games are crucial in getting players motivated to play the game. We will show that a network security game has the potential to be fun to play. Also, other interesting elements are identified based on the ideas built in this study.

An agent-based simulation model for simulating networks and attacks was designed and implemented. A game was developed designed to use this simulation model for interactive real-time network security gaming. The model involves a complex network simulation using a sub-set of the TCP/IP stack of protocols. It will be described how a complex state structure can be utilized to simulate malicious code attacks and other attacks associated with inter-connected networks.

3 Review of state of the art

In this chapter existing theories and state-of-the-art in the area of network security, gaming and gaming theory will be discussed.

3.1 Popular Simulation Games

The following chapters will describe some popular commercial games using game play elements that could be adapted to a network security context. In this study focus will be on simulation games. Computer networks are, by nature, good candidates for simulation techniques, and popular simulation games like Sim City and Roller Coaster Tycoon will be used as sources of game play ideas.

3.1.1 Sim City

Sim City, created by Will Wright, is one of the most popular simulation games to date. The first version released in 1989 sparked a whole series of simulation games.

The purpose of Sim City is to build a functional town with industrial, commercial and residential areas, and provide transport, roads, electricity and other services requested by its citizens.

Figure 1: Sim City screen shot

In Sim City the player takes the role of a city mayor and through resource management and budgets, builds the city by laying out building areas, roads and transports in a city map. The city design must be prepared for different types of disasters like riots, flooding, lightning, fire, tornadoes and Godzilla-attacks(!). The impact of disasters depends on how much is spent on fire and police services and where these services are located. How things are constructed will have an impact on different statistics like crime, pollution and traffic jams. These statistics provide feedback to the player on how well (s)he is playing the game.

3.1.2 RollerCoaster Tycoon

RollerCoaster Tycoon is another popular simulation game developed by Chris Sawyer and published by Hasbro Interactive.

Figure 2: RollerCoaster Tycoon screen shot

In this game the player is the administrator of a theme park. The objectives is to build the park's attractions and the player is rated by the visitors' happiness level. Visitors have different preferences on what they want to ride, and the player must carefully monitor emotions expressed by these visitors in order to satisfy their needs.

For this study, it is interesting to see how user preferences and demands have been implemented to provide feedback.

3.1.3 CyberCIEGE

This study is not the first attempt to build a security game. Researchers at Center for Information Systems Security Studies and Research (CISR) have developed an information assurance teaching tool called CyberCIEGE [1].

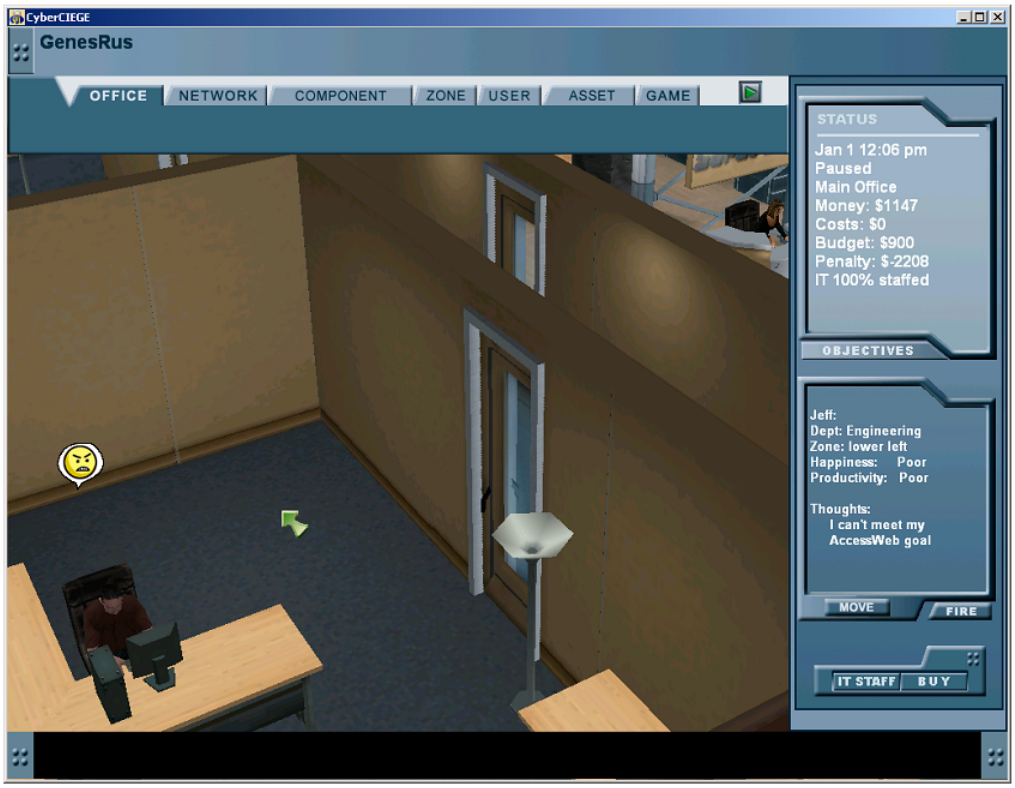

Figure 3: CyberCIEGE Screen shot

CyberCiege is a scenario-based simulation and resource management game where game play ideas also have been borrowed from games like SimCity and Rollercoaster Tycoon. Even though CC is classified as a game CC has a strong focus on learning. The approach used in this project is slightly different, were focus is mainly on how to build an enjoyable game; a game that is fun to play.

In [2], Irvine et. al, presents CyberCIEGE as “an innovative computer-based tool to teach information assurance concepts”.

CC is built on an advanced scenario definition language expressing security-related risk management tradeoffs in different scenarios. The scenario definition language includes the following major elements;

- **Assets**. Information of some value for the enterprise, like secret formulas, corporate accounting, business plans, expense statements, and marketing material.
- **Users**. Virtual users whose productive work makes money for the enterprise.
- **Zones**. Physical zones that can be used to control the physical movements of users.
- **Conditions and triggers**. The scenario designer defines conditions to be assessed by the engine during play, and specifies actions to occur as the result of a combination of condititions.
- **Objectives and Phases**. Scenarios can be divided into several phases, each consisting of one or more objectives.

Some of the ideas found in CyberCIEGE will be used in this study. CyberCIEGE involves network construction, application configuration and users who uses networks and creates assets through the use of applications.

CyberCIEGE includes both defender- and attack- scenarios.

3.2 Games and Simulations

Simulation, from the latin word simulare, means to “fake” or to “replicate”. Games usually tries to simulate or replicate real situations.

3.2.1 Distinguishing Games and Simulation Games from Simulators

Most games contains some sort of simulation. Narayanasamy’s taxonomy [3] distinguishes games from simulation games on the goal-oriented characteristics. According to this taxonomy, simulation games does not have any obvious end-state, while “ordinary” games have. Narayanasamy et. al describes what they mean is the correct use of the terms *games*, *simulation games* and *simulators*. Their article “provide a user’s and designer’s perspective on a definitive comparison of the similarities and differences between games in general, simulation games, and simulators”. A method is introduced on how to distinguish games and simulation games from simulators, using a taxonomy of observable design characteristics (fig. 4).

Identifying Characteristics		*Games*	*Simulation Games*	*Training Simulators*
1.	*Involves simulation*	1. A virtual environment is present. 2. The application interactively engages the user in a form of simulation.		
2.	*Imaginative experience*	1. May provide an imaginative or fictitious simulated environment		1. Only provides recreations of real-world environment
3.	*Entertaining, fun and engaging*	1. Provides entertainment. 2. Provides interesting & engaging challenges. 3. Provides a fun experience		1. Not intended to be entertaining, fun or engaging. 2. Operator may possibly find the application entertaining, fun and engaging.
4.	*Skills development*	1. Does not provide an application-specific skill development. 2. Possible, although not as a primary feature.		1. Operator skill-development is the primary purpose of a simulator.
5.	*Type of challenge*	1. Ideally, a continuous and intelligent challenge.		1. Challenges depicted accurately with respect to an equivalent real-world scenario.
6	*Gestalt*	1. Presence of game-play patterns. 2. Game-play patterns may vary. 3. Possible development of a game-play gestalt.		1. Presence of standard operational procedures. 2. Procedures do not change.
7.	*Goal-oriented*	1. Goal oriented activity present.		1. Goal oriented activity absent. 2. No obvious end-state.
		2. End-state present.	2. No obvious end-state	

Figure 4: Design characteristics of games, simulation games and simulators

Simulation games are classified using four categories;

- **Participatory**. Involves physical simulation and virtual/augmented reality environments.

- **Iterative**. Uses in-game parameters to derive meaningful real-world results. Players can iteratively provide input, obtain results and reflect upon the result to provide more meaningful input.
- **Procedural**. Player provides new input iteratively, reflects upon the result and provide more input. Players are encouraged to make use of predefined procedures to complete the game.
- **Situational**. A complex scenario is presented which depends on actions taken by multiple human and/or intelligent computer players.

The common denominator for these categories of simulation games is that an action-consequence model is noticeable in the game.

Simulators are commonly classified by the simulation model they adopt;

- **Monte Carlo**. Makes use of state sequencing.
- **Discrete Event**. State changes at discrete points in time when some event occurs.
- **Continuous**. State changes occur at continuous points in time.
- **Combined**. A combination of discrete and continuous.
- **Agent-based simulation**. A special type of discrete simulation where individual entities are represented directly and possesses an internal state and set of behaviors or rules which determine how the agent's state is updated from one time-step to the next.
- **Hybrid**. Uses an analytical sub-model within a discrete event model.

Narayanasamy argues that "simulation games can be defined as a mixture of a game of skill, chance and strategy that results in the simulation of a complex structure". This definition fits well in to one of the goals for this study; to simulate a complex structure of networks and network components.

Even if there are downsides in using simulations [4], a simulation fit well into a network and network security context.

3.2.2 Classification of Information Security Simulators

A game about network security should include simulating networks and security phenomenons. Saunders [5] divides simulators into different categories;

- **Packet Wars**. Packet Wars simulators are real, but isolated, networks with the only purpose to simulate network attacks and defenses at a technological level. The pros of these simulators are the use of real hardware and software. The cons would be the time and effort you need to setup a new simulation session. Hardware would need to be reset and software has to be re-installed before each new session. The cost involved in having such equipment for simulation purposes only, is of course another argument. Examples of such simulation networks are IWAR (USMA/West Point), ID'ed net (DEFCON) and Root wars (Toorcon).
- **Sniffers + Network Design Tools**. These are packages used by system administrators and system developers who need detailed understanding of packet flows and buffer overflows. Network Modeling and Simulation (NMS) packages can be used for in-depth analysis and testing at bit-level. These are very fast simulation techniques, but

only for technical purposes. Examples of such packages are Cnet, EcoPredictor, IT DecisionGuru, NetCracker and NetRule.

- **Canned Attack/Defend Scenarios**. These simulators are usually standalone game applications used as a gaming and learning tool. Scenarios vary from hard- coded attack trees using random events to more complex dynamic simulations. Examples of applications are InfoChess, CyberProtect and The Information Security War Gaming System (ISWGS).
- **Management Flight Simulators**. MFS are typically built upon a system dynamics model created to help project managers and directors to better understand the interaction between people, equipment and economics where processes influences each other dynamically in different ways. Numerous system dynamics models have been built for specific information security tasks, like the Information Technology Organization Flight Simulator, the Synthetic Environments for Advanced Simulations (SEAS, now CyberMBA). Other tools related to this type of modeling and simulation would be Matlab/Simulink and Easel.
- **Role Playing**. Finally, Saunders includes the role playing type simulators. These are often scenario based training events played out using white boards, paper and pencil. These role playing games often focus on the management role in an organization usually at a national level.

3.3 Network Simulators

In addition to IS simulators there exists a wide range of network simulators. Network simulators have been developed for different purposes, most aimed at technical analysis of packet transmission. Some are free for academic use, and others aimed at the commercial market. Network simulations often referred in the literature are;

- **OPNET**. Commercial product suite used to design, deploy, and manage network infrastructure, including equipment and network enabled applications. OPNet's modeler is actually a development tool used to analyze protocols and application in distributed networks.
- **OMNet++**. OMNET is typically used for computer networks and traffic modeling, and modeling communication protocols and distributed systems. It is freely distributed under an academic public license.
- **cnet**. This simulation is primarily aimed at networks courses and help students to better understand the purposes of the different layers of the OSI-model. cnet is designed as a discrete event simulator.
- **ns-2**. Ns is a discrete event simulator targeted at networking research. Ns provides support for simulation of TCP, routing, and multicast protocols over wired and wireless (local and satellite) networks.

3.4 Network Design Principles

In [6], Saltzer and Schroeder defines basic principles of secure design which to some extent can be used in the design of computer networks. Definitions are taken from Bishop[7].

- **The principle of Least Privilege**. A subject should be given only those privileges that it needs in order to complete its task.
- **The principle of Fail-safe Defaults**. Unless a subject is given explicit access to an object, it should be denied access to that object.
- **The principle of Economy of Mechanism**. Security mechanisms should be as simple as possible.
- **The principle of Least Common Mechanism**. Mechanisms used to access resources should not be shared.
- **The principle of Complete Mediation**. All accesses to objects should be checked to ensure that they are allowed.
- **The principle of Separation of Privilege**. A system should not grant permission based on a single condition.
- **The principle of Open Design**. The security of a mechanism should not depend on the secrecy of its design or implementation.
- **The principle of Psychological Acceptability**. Security mechanisms should not make the resource more difficult to access than if the security mechanisms were not present.

3.5 Engineering principles for IT security

In a recommendation [8] from the National Institute of Standards and Technology (NIST), 33 design principles are discussed in relevance to the life-cycles of systems;

- Initiation phase
- Development/Acquisition phase
- Implementation phase
- Operation/Maintenance phase
- Disposal phase

Many of these principles are relevant in securing computer networks and should be taken into account in this study.

3.6 Models, Metrics and Taxonomies

Best-practices in information security, standards and regulations will be considered. These includes;

- "Guidelines for the Security of Information Systems" developed, endorsed, and published by the Organization for Economic Cooperation and Development. 1992. See http://www.oecd.org//dsti/sti/it/secur/prod/e_secur.htm
- "BS 7799 - Information Security Management" from the United Kingdom Department of Trade and Industry, published under authority of the British Standards Institute. 1995 - 1999. See: http://www.bsi.org.uk/disc/
- "ISO/IEC 17799 - Information Technology - Code of practice for information security management" [9] from International Standard.
- "Control Objectives for Information and Related Technology" (CobiT) developed, en-

dorsed and published by the Information Systems Audit and Control Foundation. 1995 - 1999. http://www.isaca.org/cobit.htm

- "BS 15000 / ISO 20000 - ITIL & Incident Management" [10]

In addition, metrics and taxonomies considered to be best practice are available from the literature. These include Landwehr's taxonomy of Computer Program Security Flaws [11], the Common Criteria specification [12] and Weaver's Taxonomy of Computer Worms [13].

3.7 Game Play Theory

Theories has been developed to identify what makes games fun to play and these will be used in discussions and to test the enjoyment hypothesis.

3.7.1 What makes games fun to play?

In [14], Malone describes how to make games fun to play. His intuitions are discussed in different categories. This is an extract from his paper;

- **Challenge**. In order for at computer game to be challenging, it must provide a goal whose attainment is uncertain. A number of important consequences follow from this simple principle;
  - Goal. Simple games should provide an obvious goal. Usually the more obvious and compelling the goal is, the better. Complex environments without built-in goals should be structured so that users will be able to easily generate goals of appropriate difficulty. The best goals are often practical or fantasy goals rather than simply goals of using a skill. Players must be able to tell whether they are getting closer to the goal.
  - Uncertain outcome. A game is usually boring if the player is either certain to win or certain to loose. Malone argues four ways to make an outcome uncertain;
    - *Variable difficulty level*. Difficulties can be variable depending on how well the player plays. Levels can be chosen by the player or determined by an opponent's skill.
    - *Multiple level goals*. Multiple goals can be created by having a basic goal of accomplishing something in the game and then meta-goals of reaching the basic goal efficiently. Malone mentions score-keeping and speed-responses as means of meta-goals.
    - *Hidden information*. The game should selectively reveal information which could provoke the player's curiosity.
    - *Randomness*. Uncertain outcome could be accomplished through the use of randomness.
  - Self-esteem. Success in a computer game, like success in challenging activities, can make people feel better about themselves. The opposite is failure, which can lower a persons self-esteem and decrease the player's desire to play the game. This negative effect could be reduced by introducing variable difficulties.
- **Fantasy**.

  - Intrinsic and extrinsic fantasies. Fantasies are associated with the goal of the game and the skills used to reach it. Malone describes how intrinsic fantasies can be more fun than extrinsic ones. With extrinsic fantasies the fantasy depends on the skill, but not vice versa. Intrinsic fantasies provides feedback so that the player can improve his/her skills. For example, that consequences of the players actions are visualized (feedback) and the player can improve skills to get better results in the next attempt. Malone argues that this is more interesting and instructional than extrinsic fantasies.
  - Emotional aspects of fantasy. Malone argues that it is difficult to know what emotional needs people have and how these might (partially) be met by computer games. But he assumes that games that embody emotionally-involving fantasies like war, destruction, and competition are likely to be more popular than those with less emotional fantasies. Because peoples fantasy preferences vary, a computer game with different fantasies could have a broader appeal.
- **Curiosity**. Curiosity is the motivation to learn. Malone distinguishes between sensory and cognitive curiosity.
  - Sensory curiosity. Computer games can appeal to the sensory curiosity by using audio and visual effects as decoration, to enhance fantasy, as reward or as a representation of the system.
  - Cognitive curiosity. Cognitive curiosity can be thought of as a desire to bring better "form" to one's knowledge structures (completeness, consistency and parsimony). This can be in form of informative feedbacks. To engage a learner's curiosity, feedback should be surprising. To be educational, feedback should be constructive.

3.7.2 Flow and GameFlow

In [15], Sweetser and Wyeth constructed a model called GameFlow based on the widely accepted Flow model introduced by Csikszentmihalyi [16] who stated that

"Flow is an experience so gratifying that people are willing to do it for its own sake, with little concern for what they will get out of it, even if it is difficult or dangerous".

Flow experiences consist of eight elements;

1. a task can be completed;
2. the ability to concentrate on a task;
3. that concentration is possible because the task has clear goals;
4. that concentration is possible because the task provides immediate feedback;
5. the ability to exercise a sense of control over actions;
6. a deep but effortless involvement that removes awareness of the frustrations of everyday life;
7. concern for self disappears, but sense of self emerges stronger afterwards; and
8. the sense of the duration of time is altered.

The GameFlow criteria for player enjoyment in games;

- **Concentration**. Games should require concentration and the player should be able to concentrate on the game;
  - games should provide a lot of stimuli from different sources
  - games must provide stimuli that are worth attending to
  - games should quickly grab the players' attention and maintain their focus throughout the game
  - players should not be burdened with tasks that don't feel important
  - games should have a high workload, while still being appropriate for the players' perceptual, cognitive, and memory limits
  - players should not be distracted from tasks that they want or need to concentrate on
- **Challenge**. Games should be sufficiently challenging and match the player's skill level;
  - challenges in games must match the players' skill levels
  - games should provide different levels of challenge for different players
  - the level of challenge should increase as the player progresses through the game and increases their skill level
  - games should provide new challenges at an appropriate pace
- **Player Skills**. Games must support player skill development and mastery.
  - players should be able to start playing the game without reading the manual
  - learning the game should not be boring, but be part of the fun
  - games should include on-line help so players don't need to exit the game
  - players should be taught to play the game through tutorials or initial levels that feel like playing the game
  - games should increase the players' skills at an appropriate pace as they progress through the game
  - players should be rewarded appropriately for their effort and skill development
  - game interfaces and mechanics should be easy to learn and use
- **Control**. Players should feel a sense of control over their actions in the game;
  - players should feel a sense of control over their characters or units and their movements and interactions in the game world
  - players should feel a sense of control over the game interface and input devices
  - players should feel a sense of control over the game shell (starting, stopping, saving, etc.)
  - players should not be able to make errors that are detrimental to the game and should be supported in recovering from errors
  - players should feel a sense of control and impact onto the game world (like their

actions matter and the y are shaping the game world)

  - players should feel a sense of control over the actions that they take and the strategies that they use and that they are free to play the game the way that they want (not simply discovering actions and strategies planned by the game developers)

- **Clear Goals**. Games should provide the player with clear goals at appropriate times;
  - overriding goals should be clear and presented early
  - intermediate goals should be clear and presented at appropriate times
- **Feedback**. Players must receive appropriate feedback at appropriate times;
  - players should receive feedback on progress toward their goals
  - players should receive immediate feedback on their actions
  - players should always know their status or score
- **Immersion**. Players should experience deep but effortless involvement in the game;
  - players should become less aware of their surroundings
  - players should become less self-aware and less worried about everyday life or self
  - players should experience an altered sense of time
  - players should feel emotionally involved in the game
  - players should feel viscerally involved in the game
- **Social Interaction**. Games should support and create opportunities for social interaction;
  - games should support competition and cooperation between players
  - games should support social interaction between players (chat, etc.)
  - games should support social communities inside and outside the game

In their study it was concluded that the GameFlow model could be used in its current form to review games. Therefore, this model will be used to test the enjoyment hypothesis (fun/not fun) for the game prototype built in this study.

3.7.3 Other Game Play Theories

Irvine shares hers ideas on game play and information security in [1] and [2] which seems to be motivated by her work on CyberCiege. Also, Näkros [17] has some ideas on how to visualize security aspects using computer games as a complement to the conventional linear instruction. Their thoughts and insights into this topic will be taken into account. Other theories about game play and game content could be borrowed from Prensky[18] and Schneier[19].

3.8 Software Development

In the process of designing and developing a software product, some engineering tools are required.

3.8.1 Design Tools

A modeling tool will be needed to analyze objects, design the simulation model and the front-end for the game. The Unified Modeling Language (UML) [20] is currently considered best-practice in object oriented analyze and design and will be used in the design research. One meta goal is to design a software model that can be extended for future needs. Therefore, the design will be based on the theories of *Design Patterns* [21], a topic associated with object oriented model designs. Design patterns can be used to find the best object oriented design which in turn makes it easier to achieve features like extendibility, good performance, and scalability.

3.8.2 Development Tools

Simulations normally consumes more resources than other applications and it will be necessary to choose tools that achieves high performance, visualization and scalability. The latter is concerned with the number of entities the game will be able to simulate.

Microsoft .NET offers a feature-rich framework for software development. The simulation model, the theoretical framework, and user interface, will therefore be implemented using the .NET framework and the C# programming language. C# and .NET is widely used and is a convenient rapid application development environment for multi-tiered applications and offers numerous application layer choices. Performance issues are expected to influence the choices made in the model design.

4 Method

To answer the research questions, the following methods was used;

- *literature review and content analysis* to establish the conceptual model,
- *design research* to design the model and build the game prototype, and finally;
- *user trails* was arranged along with *interviews* using a *feedback survey* to evaluate the game.

4.1 Literature Review and Content Analysis

To establish the conceptual game model and its limitations, literature and theories on gaming and enjoyment was reviewed along with content analysis of existing, popular, simulation games.

The idea was that a network security game should be built on a complex network simulation and it should be possible to construct and configure networks and network components while the simulation was running. Which tasks in the network security domain that was suitable to create an enjoyable game had to be investigated.

Conceptual ideas are described in chapter 5.

4.2 Design Research

To build the simulation model and game prototype, based on the conceptual ideas, additional information from the literature related to network design, security and engineering techniques was used. This also included material on object oriented analysis and design, and programming techniques.

The design research methodology described in Owen's paper[22] was used the design phase of this study to ensure that the prototype would have the quality needed to simulate networks and network attacks. Research was conducted in two realms; the realm of theory (design) and the realm of practice (development). This process of knowledge building is illustrated in fig. 5 and 6.

The design research process and game model is described in chapter 6.

4.3 User Trails, Interviews and Feedback questionnaires

To evaluate our game prototype we arranged user trails and created a feedback questionnaire. This questionnaire can be found in appendix A.

Participants was selected among a group of people identified as competent in network administration and information security. The motivation behind this experiment was to receive feedback related to the game as a phenomenon. Personal interviews and telephone interviews were arranged using the questionnaire to create a structured interview.

This experiment is described in chapter 7.

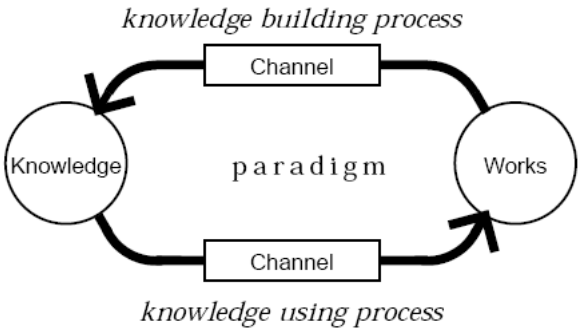

Figure 5: A general model for generating and accumulating knowledge

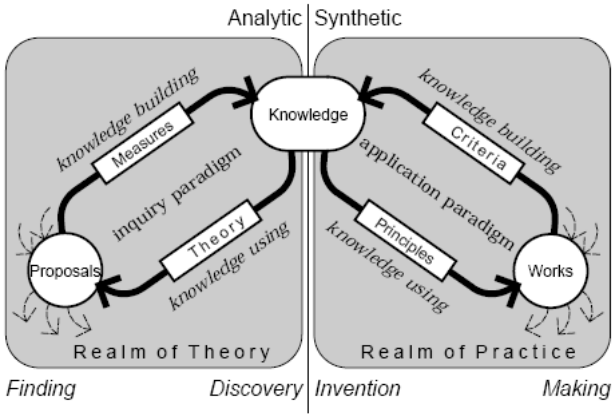

Figure 6: Using and accumulating knowledge in two realms

5 The Game

The purpose of this chapter is to come up with some conceptual ideas on how to build a network security game. This will be conducted through literature review and content analysis of existing games.

5.1 Introduction

Information security topics typically involve, or depends on the use of networks and distributed services. If a ultimate goal would be to construct a game that covers most areas of information security, the intuition is that such a game should be built on top of an existing network security game to get the best real-life experience as possible.

Networks are constructed to exchange data between hosts in an efficient and secure manner. The type of data being transferred and the need for confidentiality, integrity and availability usually depends on the businesses being conducted. Different businesses have different goals, and both security and functionality goals must be met. Businesses strive to balance these goals as requirements and threats changes over time. A set of standards has been developed to help companies define and revise security policies [9] [10] and equipment security needs [12].

Increased functionality often leads to more complex systems and system states, and with increased complexity, systems gets more vulnerable for security attacks.

In the following chapters the scope of the game will be defined. Then ideas will be discussed in relation to theories on fun and enjoyment in games. Before describing the scope there will be a short review of popular simulation games.

5.2 Game Review

To get get help in defining the concept, games like SimCity, Rollercoaster Tycoon and CyberCIEGE was used as sources of ideas.

5.2.1 SimCity

SimCity is a simulation game where the player is the urban-planner and mayor of a city. Initially, a piece of land and some money is assigned. Objectives are to use this money to build a functional city, attract citizens, and keep them happy. The more citizens the player attracts, the more income (s)he will get from taxes. As a Mayor one must try to balance the budget by keeping the taxes as low as possible but still have enough money to support public services and money to extend the city by constructing new town areas.

The player builds a city by laying out areas for residential, commercial and industrial buildings. One can also construct buildings with special purposes, like police stations, fire stations, hospitals, harbors, airports, sport arenas and by using other land marks, revealed as artifacts as the simulation progresses.

Transport is important for the well-being of the city. Roads and train tracks must be planned carefully. In later versions even underground subways and water supply can be added. Cities would need landfill areas to dump garbage and power stations to provide electricity.

SimCity has no obvious end-state but contains meta-goals like keeping citizens happy [1]. The player will have to save the city from exploding nuclear plants, pollution from coal plants, flooding, etcetera. Health care, fire and police services contribute to risk management. The cost of founding public services depends on how many of these services the player builds. The services' physical location has an impact on how citizens are attracted to different areas of the city. High crime rates and pollution leads to urban decay and lower tax-income in affected areas.

5.2.2 Rollercoaster Tycoon

In this game the player's role is to manage a theme park. Instead of constructing buildings, the player must build attractions and other facilities typically associated with theme parks. The objectives are to attract visitors, build rides and attractions in accordance to visitor preferences while maintaining the park. As they get used to different rides, visitors tend to demand more thrilling rides. The player will have to hire janitors to keep the park fresh and clean and enough engineers must be hired so that rides can be repaired and controlled at frequent, and safe, intervals. Failing to do so can result in break-downs and accidents that claim lives which, of course, will have a very negative economical impact. Being a manager, the player can also use money on advertising and offer free-tickets to attract more visitors. Entertainers can be hired to keep people happy while queuing to get onto the rides.

RCT is scenario-based, which means that the player is introduced to pre-configured game scenarios (game levels) in which defined objectives have to met in order to get to the next level.

An interesting element in RCT is it's use of individual emotions. Each visitor has personal characteristics and ride preferences derived from these characteristics. The player's failure or success in satisfying visitor demands will result in very detailed expressions and display of emotions. User statements can be expressed individually, or based on a group of visitors. These expressions gives valuable feedback to the player on how to improve the park. Feedbacks can be something like "Rollercoaster <ride name> was really cool", "The ice-cream was expensive", "I don't want to ride <ride name>. It looks too dangerous", etcetera.

5.2.3 CyberCIEGE

Inspired by SimCity and RollerCoaster Tycoon, in CyberCIEGE the player takes the role of a decision maker for an IT-dependent organization.

Scenarios are targeted for special audiences with different assurance needs, but the main objectives are to keep virtual IT-users happy and productive and at the same time provide the security measures needed to protect valuable organizational information assets [2].

Resource management is accomplished through the use of a budgets from which investments must be made to meet certain security goals. The player must make choices regarding procedural, technical and physical security. Poor choices often results in disasters. CyberCIEGE uses the potential tension between strong security and user productivity to illustrate that many security choices are an exercise in risk management.

[1] In later versions of the game, scenarios was introduced to establish main goals

5.3 Game theory

The reviewed games, and the game idea for this study will now be discusses in the context of Malone's intuitions [14] and the GameFlow model[15].

5.3.1 Challenge

. Malone[14] stated that *"in order for a computer game to be challenging, it must provide a goal whose attainment is uncertain"* and *"In a sense, the very notion of **game** implies there is an **object of the game**"*.

With SimCity it became obvious that games did not always need to define clear goals to be popular among players. This surprised the gaming industry, which by then was convinced that games needed clear goals to be attractive. Even if later versions of SimCity introduced city scenarios where the player had to keep some statistical game parameters (crime, pollution, etcetera) below or above given thresholds to succeed, the basic game idea was to build a city and keep citizens happy. Typically, people could play for hours fascinated by the simulation and visual animations. Goals was defined by the player's themselves and they could try different strategies, like building the largest, cleanest, most polluted city, etcetera.

In RollerCoaster Tycoon, scenarios have different goals that must be met for the player to succeed. This could for example be to attract a given number of visitors or reach a given park value within a defined time period. Scenarios can have more than one goal on which some meta goals or all goals must be reached successfully. Reaching a scenario's main objectives is typically rewarded with new and more complex scenarios. There are multiple advantages to this, according to Malone. First, the player is rewarded for skillful playing which increases self-esteem. The game's difficulty level is adapted to the player's skills, and it will bring uncertain outcome into the game to provoke the player's curiosity; hidden information, like a new scenario, is revealed periodically.

Instead of ending the game, in RCT, even if the player successfully completes the main goals, (s)he can choose to continue building the park. Some scenarios are relatively fast to finish while others can be played for hours, therefore players had this option to be able to continue playing.

SimCity and RollerCoaster Tycoon as well as other simulation games present the player with a complex environment, a city or theme park, and it is the player's task to construct or maintain this complex structure of entities.

Malone states that

"complex environments without built-in goals should be structured so that users will be able to easily generate goals of appropriate difficulty". Most people like to construct things, make plans, create hypothesis and test them [8].

Sim City and Roller Coaster Tycoon are both simulation games with no obvious end-states, but scenarios has been defined to create environments with different goals. This approach should be considered for this study. For instance, by defining different networked scenarios and environment characteristics with preset success and failure definitions.

5.3.2 Fantasy

Malone uses the term *fantasy* to illustrate the game setting and how players are able to identify with this setting. Because people's fantasy preferences vary, a computer game

with different kinds of fantasies could have a broader appeal.

For SC and RCT, it is an advantage that most players already are familiar with cities, and theme parks, and it is therefore easy most people to adapt to the fantasy of building a city or constructing roller coasters in a theme park. In later versions of RCT, it is even possible for players to become the visitor of a theme park and ride the roller coaster they have created using in-game virtual reality ride sequences. This extends the fantasy even further.

It is probably not just as easy to adapt to the fantasy of being a network administrator, and it should be investigated how to enhance the fantasy or make the game more appealing by other means.

5.3.3 Curiosity

Sensory curiosity relates to decoration and the use of graphics and sounds to enhance the fantasy. In SC, one can hear cars making sounds, people shouting in riots, explosions and other sound related to events that occur in the game. In RCT one can hear people chatting very life-like and in a recognizable way for people who have visited theme parks. The graphical representation is also nicely selected; a city and a theme park seen from a bird view that includes graphical elements and animations coupled with the fantasy. Graphical elements that could be used in a network security game could be;

- a network map showing how different hardware equipment are connected,
- equipment power status and traffic flow indicators (network card lights),
- virtual users and their happiness level,
- newsflashes when something important happens, like when new viruses are detected
- traffic-sniffing information showing packet traffic
- textual description of actions performed by virtual users along with user complaints and compliments.

These effects will also act as feedback mechanisms that improves the cognitive sensory.

It is harder to identify what kind of sound effects and background music one could use[2]. Sound effects could be;

- different vignettes being played when computers are powered on and off,
- a pool of sounds including angry outbursts when virtual users gets angry or cheering sounds when they become happy,
- sound effects from anti-virus applications when a virus is found. This could be complemented with a graphical animation of the host being infected

Sounds should be exaggerated and include humor. The same applies to graphics. In RCT it is quite interesting, and for morbid players quite funny, to observe details like when the janitor must clean up vomit from visitors who have been sick from riding thrilling attractions. A way to minimize this problem is to place ice-bars, hot-dog stations, or other food facilities far away from these rides. It is also a good idea to place a public toilet close to the roller coaster's exit-area.

[2]the one that immediately comes to mind is quite annoying and is probably not something that would make the game more attractive

Curiosity is closely related to the fantasy, and by combining nicely adapted graphics and sound effects with humor, one could enhance both fantasy and curiosity.

5.3.4 Ideas from GameFlow theory

It should be possible to complete a task. Game scenarios should be designed so that players will always have the opportunity to finish the game successfully. "Acts of God" events should be eliminated or kept to a minimum. A countermeasure should exist for all events that occur in the game. Introducing random catastrophic events into the game is therefore not a good idea unless the player have means to prevent these events from happening in the first place.

The game should be constructed so that players feel a sense of control over actions. The graphical interface must be easy to use. Tedious tasks should be generalized so that actions can be performed on multiple hosts simultaneously using template techniques and bulk-operations. For instance, when new applications are installed, the player should be able to install on multiple computers at the same time. Performing the same action on every host in a large network would be considered boring. An example of generalization in the game interface can be seen in SC. Instead of constructing buildings, one bye one, areas for different building types can be laid out as city blocks. This makes it easier and faster to build large cities.

Another idea from GameFlow is to introduce social interactions. It should be possible to compete or cooperate with other players. This is absent in RCT and SC due to the nature of these games, but this idea should fit well into the cooperating nature of computer networks.

Competition can be accomplished by different means;

- by having public scoreboards where players can compete on achieving the best high-scores, and/or;
- real-time competition where players compete by trying to destroy what other has built or steal other player's assets, and/or;
- real-time cooperation where players build alliances with other players and cooperate in attacking their common enemy players.

In a network security game, the simulation model should be complex enough so that players can use different means, like crafting network packets, building probe-applications and viruses/exploits to attack other sites.

A multi-player game requires a central server to which players can connect to join other players. Such a server could act as an Internet node and distribute packets between local sites managed by each player. Using real networks to distribute packets between networks could also increase packet traffic diversity and randomness, hence enhancing the simulation realism.

5.4 Game idea

To create a game about network security, it seems natural to place the player in a setting where (s)he plays the role of a network administrator who must defend the network infrastructure, including assets, from attacks. An important task, being a network administrator or network security manager, is to plan and construct the network architecture by defining its layout and properties. A network map with connected nodes and easy access

to configuration data is therefore probably the best way to construct the game interface. Figure 7 shows the first graphical draft of the game interface idea.

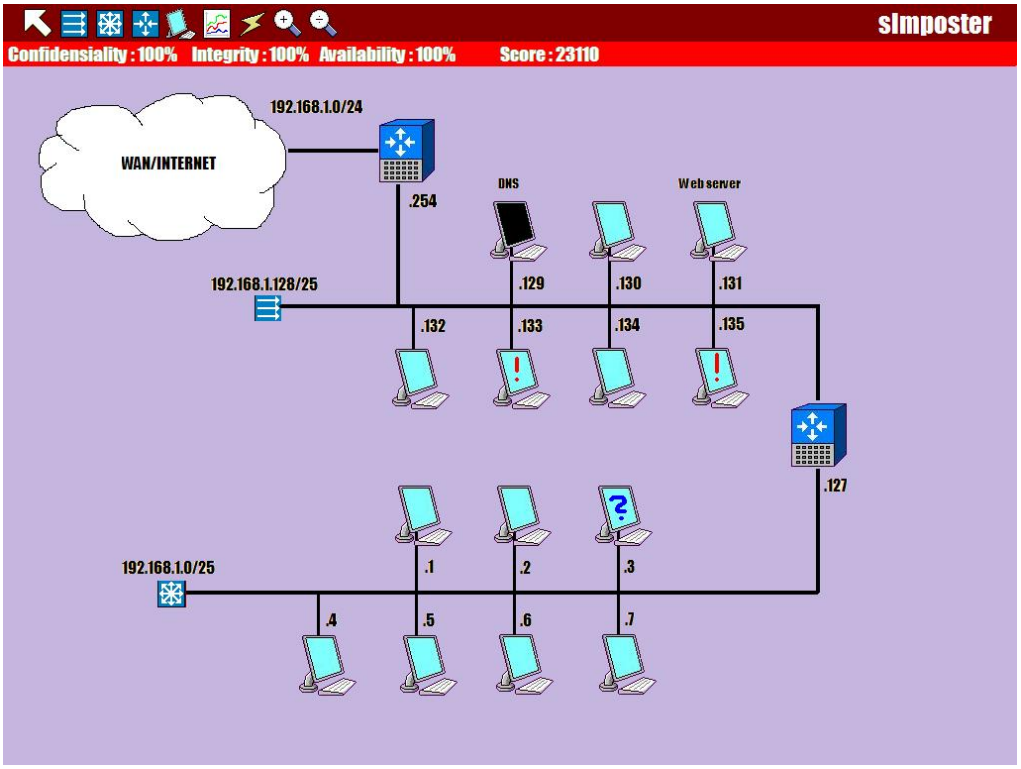

Figure 7: First conceptual draft

5.5 Scope

To be able to create a game with substance, more than just networks are needed. Network traffic is also required. To create traffic, physical and logical communication end-points must be implemented. In practice, elements in all layers defined in the Open System Interconnection Reference Model (OSI Model) needs to be involved.

Layer	Name
7	Application Layer
6	Presentation Layer
5	Session Layer
4	Transport Layer
3	Network Layer
2	Data Link Layer
1	Physical Layer

Table 3: The OSI model.

5.6 Internet Sites

The Internet should be an essential element of the game. Attacks will typically have its origin on the Internet and the player must take control of a site connected to the Internet.

To separate the player-controlled site from other sites connected to the Internet, from now on the player-controlled site will be referred to as the *local site*. Other sites will be referred to as *remote sites*.

To enhance the simulation realism, remote sites should be simulated in the same way as the local site. It should be possible to have remote sites created and removed dynamically during the game, because the player should not be able to build a complete knowledge on which domains and IP-ranges are trustworthy and which are not. A large amount of remote sites should be simulated.

Using ideas from reviewed games, a network security game should include features like resource management, network design and configuration, host and application configuration and risk management. A wide variety of attacks should be simulated.

5.7 Physical Network components

To simulate the OSI communication physical network components are needed;

- **ISP connections** for Internet connectivity,
- **switches and wiring** for network connectivity,
- **routers** to route packets between network segments, and;
- **hosts** as physical communication end-points.

5.8 Network traffic

To enable traffic between inter-connected networks, the TCP/IP stack of protocols should be used in communications. A subset of this stack should be implemented to support basic application protocols and traffic flow. In addition, the following items are needed;

- **sockets** as logical communication end-points,
- **applications** for traffic diversity,
- **software** for application diversity,
- **processes and threads** to execute applications, handle tasks, and manage sockets, and;
- **tasks** to start processes and threads.

5.9 Users

Without an engine, vehicle simulators are of little use. The same applies to network simulations. In a network context, the engine is the users who “drive” the network through the use of client applications, maintenance of networks, and configuration of networks and services.

To reduce the burden on the player, virtual users should use application (agents) while maintenance and configuration tasks should be handled by player.

5.10 Resource management

If a company had unlimited resources, then security and functionality goals would easily be met. This is usually not the case in real life. Designing and configuring hosts and networks comes at a cost. To obtain optimal results, money spent on equipment and maintenance should not exceed the goals of protection and business needs. However,

some investments could have positive effects on productivity. These are challenges the player should be faced with. Cost is typically associated with products and maintenance, therefore equipment and software should have a purchase and installation and/or configuration cost. Two models can be used, a one-time price, or a lease cost where cost is accumulated as time passes. Cost intervals could be any discrete interval, like day, week, month or year.

5.11 Attacks

Security and risk management should be the most essential part of the game. Therefore, we need to simulate attacks and other events related to network and host security.

The purpose of attacks is to bring instability to the network's state model. The objectives of the player should be to construct and maintain a structure that is resistant to attacks and, at the same time, satisfies functionality demands.

Entities with states in the network's state model should include; memory, applications/software products, processes/threads, program files, data files, configuration files, sockets, I/O buffers, network packets, routing tables, firewall filters, address resolution tables, DNS configurations and equipment power.

Applications should be able to change the states of host by execution simulated programs. Such a feature would make it possible to construct a wide range of malicious code attacks.

The game should include attacks like;

- **Traffic attacks**. Denial of Service Attacks, SYN-flooding, etcetera.
- **Spam**. Mails that have a negative impact on productivity.
- **Viruses and Malware**. Software infected with malicious code.
- **Vulnerabilities and Exploits**. Software with vulnerabilities attacked by content that exploits these vulnerabilities.

5.12 Spreading mechanisms

Also a variety of mechanisms to spread malware and exploits are needed;

- **E-mail services**. Mail clients and SMTP and POP3 servers.
- **Web services**. HTTP clients and servers.
- **File transfer services**. FTP clients and servers.
- **Remote File Services**. Services like SAMBA.

The list could be much longer, and include applications like file-sharing, instant messaging, VoIP[3] and more. The a goal should be to design a simulation model where these applications easily could be implemented. This should be sufficient enough to create a game that includes (some of the) main security events experienced in the real world[23].

5.13 Countermeasures

All attacks should be associated with countermeasures, like;

[3]Voice over IP

- **Network topology design**. Including defense-in-depth strategies and the ability to compartmentalize the network design to increase the general security of the network.[6] [7]
- **Static routing**. The ability to route packets between sub-nets [24].
- **Static firewall configurations**. The ability to filter packets on header information [25].
- **Anti-virus applications**. The ability to detect viruses based on signatures.
- **Patching options**. The ability to repair software vulnerabilities.
- **Host hardening**. The ability to reduce the complexity of host state models and reduce the number of possible attack vectors into the system.[26]

5.14 Limitations

To limit the scope, it was decided not to include physical security and data access controls in the game. Many attacks on distributed systems are related to password attacks, attacks on access control mechanisms, equipment sabotage, theft or unauthorized physical access, but these are not events that will be included in this version of the game prototype. Later versions should be able to extend the model by involving other layers of security. Figure 8 illustrates this idea;

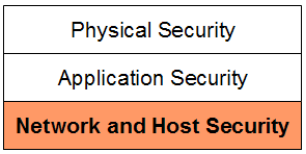

Figure 8: Security Layers

Simulated users should not try to attack systems from the inside. Even if insider attacks are common, these events are not directly related to the events in network and host security described in chapter 5.11. This decision can be seen in relation to not include access control mechanisms. To simulate insider and access-control attacks, a complicated scripting facility is probably needed in the model.

In the next chapter a design process using the conceptual ideas described in this chapter will be described. The purpose is to build a network and attack simulation model upon which a game prototype will be constructed.

6 Design Research

To construct the simulation model and game prototype the design research methodology proposed by Owen [22] was used. In the *realm of theory*, theories on TCP/IP, networking, information security, network security, viruses, exploits and enjoyment helped us build the game model. In the realm of practice, engineering techniques was applied to implement the model and test, accept and/or reject ideas. The result has been a game model[1] on which our game has been built.

To help building the model, object oriented analysis and design techniques, and design pattern theory[21] was used.

6.1 The Design Process

Game development is a large undertaking[27] which normally requires multiple roles, from the designers, graphical artists, and musicians to the project managers, developers and game testers.

Having established the conceptual ideas and the model scope, the next step was to construct the game prototype. The challenge was to implement as much as possible in five months. The total period for this study was six months, but one month was needed to conduct an experiment and write this documentation. In the following chapters, the design process will be described along with a presentation of the game model.

6.1.1 Development platform

It was decided to target the Windows 2000/XP operating system due to its wide spread use. The Microsoft .NET 2.0 framework and the Microsoft Visual Studio 2005 (VS2005) was installed on a computer with a 2.5 GHz Intel Pentium 4 processor and 512MB ram. At the time of this study this was considered a medium-end system. StarUML[2], a free graphical design tool for the Unified Modeling Language, was installed. Other recommended alternative could be Dia[3] or Rational Rose[4].

C# was selected as the programming language of choice. C# is included in the VS2005 IDE[5]. In addition to the integrated help function in VS2005 (MSDN), [28], [29], [30] and [31] was found helpful.

6.1.2 Building the executive kernel

The executive kernel[32] contains a collection of network entities and virtual users and is the heart of the simulation. This is where the simulation loop is located (ref. appendix D). The kernel is also responsible for managing other parameters associated with the simulation, like the timer, event distribution and statistic accumulator. The purpose of the first tests was to evaluate the performance of the agent-based simulation kernel. Simple network agents, routers and hosts, were implemented with a simple protocol handler to

[1] knowledge base
[2] http://www.staruml.com/
[3] http://www.gnome.org/projects/dia/
[4] http://www.rational.com/
[5] Integrated Development Environment

generate traffic. Tests confirmed that the model could manage large amounts of nodes (>1000) with good performance results indicating that a complex agent state structure could be handled by the simulation. From then on, the state model of the simulated agents was extended in iterative development cycles.

The kernel was implemented as a multi-threaded kernel. Multi-threading introduces randomness in data (traffic) flow because it is hard to predict when threads are executed and context-switches occur. A Disadvantage with multi-threading is that context-switches comes at a cost. The operating system handles the thread scheduler and performs context switching at frequent intervals. Too many threads will therefore reduce the general performance. In theory, each agent could have its own dedicated thread, but this would require too many threads. It was decided to create a pool of sixteen threads each running in separate loops. Agents are distributed randomly to different threads when the simulation starts. Some agents, like the virtual users does not require too much processing time, while hosts has a complex state model and need more processing time. If agents was not distributed randomly, one could risk that virtual users resides in one thread and hosts in another which would make the host thread execute slower than the thread with the virtual users. As the number of agents is relatively high with more than 300 remote sites and three agents pr. remote site, a random distribution will ensure that processing time is evenly distributed among agents.

6.1.3 Deadlock issues

In the realm of practice deadlocks was experienced. Deadlocks can occur in multi-threaded designs and are hard to identify due to the randomness of context-switches which makes it hard to repeat the cause of the deadlock. As the entity state-model got more complex, frequent deadlock occurred. These problems were solved by carefully placing synchronization mechanisms (monitors/locks) on resources involved in the communication between two different agents and cultivate the use of queuing patterns in the design.

6.1.4 Networking

During the first two months of the design process, focus was on modeling the networked agents and related entities used to build the agent's state structure. Literature found useful in this this part of the design was [33], [34] and [35].

A sub-set of the TCP/IP and UDP/IP stack of protocols[34] was implemented, with enough features to handle basic network traffic. This design included the routing manager/table and a a sub-set of the ARP mechanism.

It was decided not to implement IP fragmentation. The simulation should transmit data in one packet. A single communication session between two hosts could involve multiple transmissions in both directions. This decision was made to keep the model simple but still make it possible to simulate protocol sequences.

IGMP and ICMP were not implemented. This feature was postponed. With ICMP in particular, there are known probing techniques and network attack types, and its implementation should therefore be implemented in future work. The current model was designed with this in mind.

Routing was achieved by including a routing manager in host, router and firewall agents. Lower-layer nodes, like switches and WANs, uses an alternative sub-net table. A sub-net table does not include any port/interface information and its purpose is only to make it easier to distribute IP addresses when new network agents are connected to

the network segment. A free IP address from the IP-range found in the sub-net table is derived from all connected nodes and automatically assigned to the last connected agent. If multiple IP-ranges are found in the sub-net table, new connections will receive multiple free IP addresses, derived from each of the registered sub-net ranges.

Both UDP and TCP were implemented. Data associated with a protocol request, or response, is sent in one packet/datagram. Therefore, the TCP sub-set was not designed to include flow-control, window size, or sequence and acknowledgment numbers.

To activate UDP and TCP functionality, a general socket class for receive and send operations was implemented in two specialized classes, one for handling UDP functionality and another for TCP. These sockets work in close relation with the transport layer and socket manager. The TCP socket class' state model ensures that a session is set up and shut down according to rules defined in [36].

6.1.5 Building the Application Protocols

At this point, all entities needed to simulate traffic between two logical endpoints were available in the model. Several tests were conducted with dummy protocol units using UDP and TCP sockets to send and receive data between hosts. A set of topologies were constructed and saved as template files, each topology represented different hops between hosts.

A more "intelligent" and life-like application protocol generator was needed. It was identified that hosts needed entities like processes and files to simulate these protocols. Handling code execution is a complicated task in real-life systems. Different approaches were considered, and the best solution was to introduce threads as the smallest unit of simulated code execution. Server applications in particular, often use multi-threading techniques to handle multiple sessions with remote clients. Different classes were defined to accomplish simulated code execution in host agents;

- **Application**. This class contains methods and properties associated with an application in general terms[6], and defines the algorithm, the name of the application, a reference to the protocol name[7], and other application specific properties. Specialized application classes inherits from the base application class and overrides the procedure method to implement their own algorithm. This algorithm is used by the thread class to simulate the application protocol.
- **Command**. The application class has a collection of command-objects used to create application-specific tasks. Each command object has a coded string which resembles an API[8] reference including the procedure name, its parameters, an entry-point in the application algorithm and a description of the command. The description is used in feedback messages.
- **Software**. The Software class contains information about a specific software product. While the application class contains information common to all products of a specific type, the software class contains information for a specific product, like performance properties, pricing, product name, etcetera. These properties can be used by the application algorithm and are accessible through the thread-object passed in as parameter to the application algorithm. Even if the application algorithm is common

[6] web-server, mail-client, etcetera
[7] HTTP, FTP, etcetera
[8] Application Programming Interface

for all products of the same application type, it could still behave differently based on different product characteristics.

- **ProgramFile**. A program file is a host-instance of a software product. The software class can create new program files which are then added to a host's file-system. A program file has a reference to the software product. Other properties can be found in the program file, like the current patch. If the program file is infected by malware, a reference to this malware can be found in this class. Program files are involved in the creation of host processes.
- **Task**. A task is an instance of the command object found in the application class. While the command class defines a method in the application algorithm, the task holds the actual parameters for this method. Tasks are created by applications. Task parameters are calculated randomly and to some extent calculations are based on user-characteristics.
- **Process**. Processes are entities executing one or multiple instances of a program file. When a process is started, a main-thread is created and executed by the application class. Each application type can define its own startup-algorithm. For example, server applications could start listening on a specific TCP port or spawn threads that should loop infinitely while the process is running. The process class manages multiple threads responsible for executing the algorithm defined by their corresponding tasks.
- **Thread**. The thread-object contains the internal state of a thread, like the run-pointer, sleeping state, a socket reference, blocking state, the associated task, etcetera. Threads uses sockets to generate network traffic through as set of socket methods. The thread object can access all host-states, enabling these states to be changed by the application code.

Designing these classes was a complex process, but it was necessary to achieve some of the diversity and complexity seen in real-life systems. Not only are the means available to simulate application protocols, but also the capability to simulate a wide range of code-attacks intended to bring instability to the host's state model. The motivation behind the thread-execution design was that this pattern could make it easier to later convert it into a linked-list of execution-blocks. This way, it would be possible to move from hard-coded application algorithms, to dynamically defined algorithms enabling players to construct their own set of algorithms and malwares. The linked-list design was not implemented in this study, but design choices described in this chapter would make this feature easier to implement in the future.

6.1.6 Simulating Attacks

So far, works have been focused on building a stable running network simulation. Now, attacks were needed to bring instability to the system. [37], [38] and [23] were useful resources in identifying model topics related to attacks.

A malware class was defined. The purpose of this class was to extend the definition of malware tasks. A task is normally associated with application functionality and user-activity, while a malware references a task pointing to some malicious block of binary code. Malwares are referenced by infected program files. When an infected program file starts a new process, a malware thread is automatically spawned and starts executing

the malicious code in the program file's process.

A malware-factory class was designed. This class is managed by the executive kernel, and depending on the malware frequency setting in the executive kernel, new random malwares and exploits are created by the factory class in random intervals. A random trust value is calculated when new malwares are created. This random value decides how distributed the malware would be (low-spread/high-spread). The malware-factory then distributes new malwares by creating a program file with a random name and marks it as infected by setting the program-file's malware reference to the newly created malware object. Then each site is iterated and server hosts receive a copy of the infected file if the site's trust level is lower than the random trust value.

Exploits are malware but extends its definition by including references to software products and patch-ranges. A process, and its program file, is infected only if the exploit matches the associated program-file's software product and patch version. Two types of exploits were defined; file exploits and packet exploits. File exploits are distributed as data-files[9] to remote sites. Vulnerabilities in products are simulated by having exploit-tasks creating a new malware-thread in the infected software-process when a file associated with this process is opened.

Packet-exploits are simulated in the interface between the thread and the socket during send and receive operations. Packet exploits do not use files, instead the malware-factory distributes these exploits to a site's exploit-knowledge collection much in the same way it distributes infected program and data files. When users connect to a server at one of these sites, the server will randomly mark returned packets with exploits having the same protocol as the server. This resembles a packet being crafted at the remote site with the intention of exploiting vulnerabilities in client software products. If the client application matches the exploit's product and patch properties, the client process will start a new thread and execute the associated exploit task/procedure.

Server applications was also designed to have vulnerabilities. Virtual users at remote sites visit servers at the local site at frequent intervals and packets can be crafted at the remote site to exploit vulnerabilities in server applications at the local site.

In addition to malwares and exploits, an attack-tool application was constructed. Some remote sites uses this application to probe the local site and attack private services. It tries to attack all protocols known by the simulation in random sequences and at random intervals. One can never know when, from where, and against what such attacks occur. Finally, spam mails were introduced. Mails could also contain infected files in attachments.

Virtual users uses different application to download (or receive) files. Some of these files can contain malware. Different countermeasures like anti-virus applications, backup tools, firewall configurations, host-hardening, network design, etcetera must be applied by the player to prevent or reduce the impact from attacks.

6.1.7 Building the graphical front end

Building the graphical front-end was accomplished in multiple steps. Both top-down and bottom-up approaches was used in building the game model and prototype. Experiences in the front-end gave more insight in how to design the model, while changes in the model gave new ideas on how to build the front-end.

[9] image-files, word processing-files, HTML-files, etcetera

A new binary component was added for the graphical classes and functions. It was designed to poll state information from the simulation model and represent agent-states by different graphical means. Normally, a game uses OpenGL or DirectX to create the graphical representation. These libraries enhances the graphical performance but are hard and time-consuming to use. Instead, it was decided to create the graphical front-end using the GDI+[10] library included in the .NET framework. GDI+ extends the GDI capabilities of Windows but does not perform as well as OpenGL and DirectX. GDI+ does not include library functions for 3D graphics, but was sufficient enough to create the network map seen in the game. The 2D capabilities of DirectX or OpenGL could improve performance by some magnitude, and should be considered implemented in future versions. The simulation model and graphical library was built in separate components with this in mind; to make it easier to change the graphical platform if neccessary. By looking at games like CyberCIEGE and newer versions of SimCity and RollerCoaster Tycoon, one can see that these game uses 3D in their graphical representation. This seems logical, as these games strive to represent the physical world (offices, towns and theme-parks). A network map does not need to be represented in 3D.

An interesting article about the increasing complexity of building computer games can be found in [27].

Music and sound effects are considered to be important elements in creating an enjoyable game [14]. There was not enough time to include sound and it should be expected that this would have some negative impact on the game experience.

Using ideas from existing games, and GUI[11] coding standards, the main game window was designed to include a toolbar with graphical icons, general game information and an editable network map. The network map was designed to handle actions defined by the simulation model, like;

- creating and destroying network nodes,
- reorganizing and moving nodes in the network map,
- connecting and disconnecting nodes,
- changing node-states like routing information, firewall filters, installed applications, etcetera
- monitoring network-traffic and socket activities

In addition to graphical functions, the model component included classes to represent agents visually. Different approaches in creating a network map was tested, and the most effective one was a rectangular representation with nodes located in grids. The grid model made it possible to fine-tune the algorithm responsible for converting the data structure into a screen representation so that the updating speed, when the player scrolls the map, is the same regardless of the number of columns and rows used by the map.

Efforts were made to make the graphical interface easier to use and understand. One goal was to keep it as simple as possible requiring only as few mouse-clicks to access information. A right-click pop-up menu was added to make easier access to node properties and actions. A pop-up menu is illustrated in figure 9.

Shortcut keys were introduced to give easy access to frequently used game functions.

[10] Graphical Device Interface
[11] Graphical User Interface

Figure 9: Processes and Threads

By double-clicking any node, a suitable dialog box for that node type is shown. This idea was collected from RollerCoaster Tycoon.

A set of icons[12] was created for the game toolbar, network map and dialog boxes. It was important to design these icons so that they gave the user an intuitive feeling of what they represented. Some icon ideas was collected from other games and the Cisco set of "standard" icons.

6.1.8 Converting a Network Editor into a Game

At this time a network simulation, and a graphical front end able to configure it, was available. To create a game, a few more elements were needed. A game would need objectives, and a set of mechanisms to define these objectives and means to find out if they were met had to be implemented. Two types of mechanisms were implemented; statistical counters and triggers that checks if some statistical threshold has been reached. A large number of statistical counters were identified and implemented in the model where their associated events occurred. The identified counter-types (statistical enumerators) are listed in appendix C. The code sample in appendix D and E, includes code that shows how values are accumulated for some of these counter types. For each simulation loop iteration, preset triggers are compared with accompanying statistical counters. If a counter is below or above a given trigger value, the trigger will increase the value of another statistical counter, for example, the GAME_End counters. GAME_Failures and GAME_Successes can be used to count the successes and failures of meta-goals. Two statistical counter types were created to end the game; GAME_EndFailure and GAME_EndSuccess. A value other than zero in these counters will end the game.

Triggers and statistical counters are contained in a collection maintained by the executive kernel. Counters maintained by the executive are the sum of all statistics from the simulation. In addition, each agent and the local site have its own set of statistical counters. This way, it was possible to figure out how many mails was being sent from a

[12] Small graphical images

remote site to the local site by subtracting statistics on mails sent from the local site with all mails sent found in the executive statistics. In the current model, triggers are designed to target the total statistics found in the executive, but it should be possible to change this in the future so that triggers can act upon any agent and/or site.

6.1.9 Designing the Game

A game menu was created to introduce players to different game scenarios. The names of each scenario are presented as a list in the game menu as seen in fig. 10.

Figure 10: Game menu

The player starts the game by selecting any of the available game levels and pressing the play button. No restrictions were made in the order these levels could be played.

6.1.10 Brazilian Dance

The first game included a simple network connection with one host; a home computer connected to the Internet. In addition to storing infected data and program files, and crafting exploit data packets, some hostile remote sites run attack-applications that try to explore and penetrate network services at the local site. The remote file server (Samba) is one of them. The objectives are to harden the host so that remote script kiddies don't get access to the local host's remote file system through the Samba protocol. When the game starts, a dialog box is displayed and describes the game objectives along with the game characteristics. The dialog, for the Brazilian Dance level is shown in fig.11

Figure 11: The Brazilian Dance introduction dialog

The game ends if a remote attacker succeeds with more than five attacks (failure) or the player can withstand more than five attacks within fifteen simulated days (success). When the game ends, a new dialog box is shown containing the statistical results and a final score. During the game, the score increases when events considered as positive occur in the simulation. These are events associated with a functional network, like the number of mails sent, or web pages downloaded successfully, etcetera. Negative scores based on negative events are not calculated. Instead, negative results are shown in the end-game dialog box as penalty values reducing the final score. The end-game dialog box can be seen in fig.12. The motivation behind this approach was to make the game more realistic. If the player was able to notice a decrease in score during game play, it would indicate that something was wrong. But, even with this approach, a covert channel is introduced. If the player notices a slow-down in score increments, this could indicate that the network is not working as well as it should.

6.1.11 Small Office Paranoia

In this scenario, objectives are to manage a small company network with one sub-net, a firewall connected to the Internet, and two servers; one for internal services (intranet servers) and another for public services. The player will need to take extra care in selecting the most suitable applications to reduce the impact of viruses and exploits. The best virus applications are not necessarily the most cost-effective ones. Other applications,

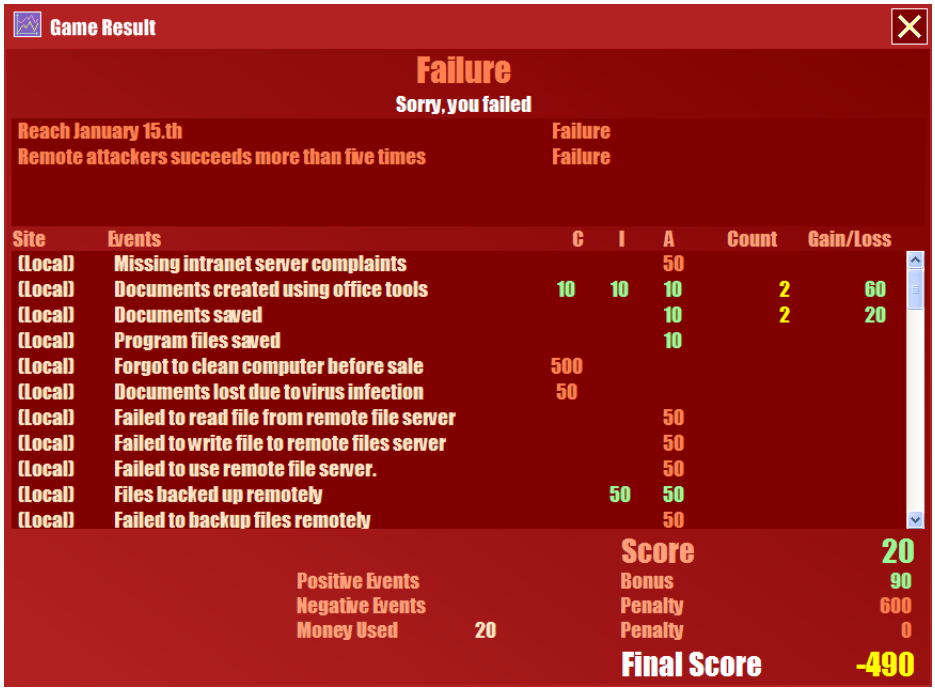

Figure 12: The game result dialog

like backup-tools can be installed to improve file integrity. A local mail server can be set up so that mail-filter applications can be installed, reducing the impact spam mails have on user productivity. As in the first scenario, the player must prevent probing and attacks on server protocols.

6.1.12 Wrong Turn

In *Wrong Turn*, the player is introduced to a network with multiple sub-nets. The objectives are much the same as in the two previous scenarios. In addition, one of the routers has a bad configuration where the default route is missing. The player must remember to, first, configure the network in a safe way before attempting to identify and fix the problem.

6.1.13 Netrepreneur

In Netrepreneur [13], the player starts with a network with one sub-net connected to the Internet. The player will need to use all his/her skills to build a functional and secure network. Employees are hired at frequent intervals forcing the player to extend the network and plan for growth. A low number of successful attacks will lead to failure.

6.1.14 Free Play

In this scenario, a network must be built from scratch and (a) Internet connection(s) must be established by the player. The objective is to build the network as large as possible. New employees (virtual users), are hired when new client-computers are purchased.

[13] Intended word-play

6.1.15 Calculation Scores

Some model configurations had to be configured using heuristics. This was related to how game scores were calculated from statistical results and simulation speed, including the distribution of events.

As mentioned in chapter 6.1.10, scores are calculated for positive events during game play, and the negative events and their score penalties are represented in the end-game dialog. A metric on how to calculate scores had to be created using heuristics. Events defined by the statistical enumerators were weighted with positive or negative values for each of the three security definitions; confidentiality (C), integrity (I) and availability (A) (see fig.12). The intention was not to create a perfect metric, but for gaming purposes it seemed to work well after a short trial and error period. In addition, this metric used importance values for C, I and A collected from the dynamic scenario settings so that score calculations also had dependencies to how the played scenario weighted the importance of these definitions.

6.1.16 Balancing the game

Having a good balance is important for the overall game play. If events occur too fast, or too slow, the player will get frustrated and lose interest[15]. A network game will probably be too boring if it was simulated in true-speed. To speed up events, simulated dates were introduced. Even if data traffic and user activity seems to be happening in real-time, the game-time is accelerated with simulated dates changing every few seconds. This idea is also seen in the reviewed games. Heuristic was used to find what was considered the best day-change interval. Intervals are static and equal for all scenarios. Simulated time relates to budgets and running costs, and are used by triggers to end the game at a given date. When the executive kernel finishes a loop, a time counter is increased, and currently, a simulated day correlates to 300 agent executions. Having established the kernel timer and number of executions per simulated day, it was easier to find out how to balance the other elements in relation to resource management, and network and user events.

Dynamic values can be configured for each scenario to fine-tune the frequency at which events occur, like how often new malwares and exploits are introduced, etcetera. By changing dynamic parameters in the executive kernel new events are introduced at different frequencies for different scenarios. Even more specific, virtual users can be preconfigured with different characteristics. A scenario can be made more challenging if virtual user's awareness and efficiency characteristics are lowered. This would result in more virus infected applications being downloaded and installed by these users.

6.2 The Game Model

In the next chapters the simulation model will be presented. This section starts with a description of the main components, and a short introduction to the UML notation.

6.2.1 General model issues

It was necessary to separate the logic dealing with the simulation, the graphical layout and front end, and a decision was made to construct three separate software components;

- **Net**. This is the inner-most module containing the executive simulation kernel, enti-

ties, statistical module and timing facilities (implemented as a DLL [14] file)

- **Net.Model**. This component wraps entities in the Net component for graphical representation purposes. It also contains a library of graphical functions used by the game component. (DLL)
- **Net.Model.Game**. The executable game component which uses the other two comoponents (EXE)

Figure 13 illustrates the components and their dependencies.

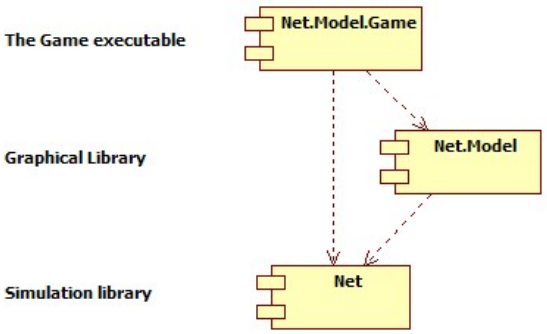

Figure 13: Component architecture

6.2.2 A short UML notation introduction

In order to better understand the notation used in the following chapters, a short introduction to UML will be given. A more detailed description can be found in [20].

UML defines several diagrams to represent a system;

- **Class Diagrams**. Describes the static structure of a system.
- **Package Diagrams**. Package diagrams organize classes into related groups to reduce the dependencies between packages.
- **Object Diagrams**. Diagrams that describes the static structure at a specific time. Objects are instances of classes.
- **Use Case Diagrams**. Models the functionality of a system using actors (roles) and use cases (activities). Use case diagrams are the “glue” all other diagrams must relate to.
- **Sequence Diagrams**. Describes the interaction among classes in terms of exchange of messages over time.
- **Collaboration Diagrams**. Represents interactions between objects as a series of sequenced messages.
- **State Diagrams**. Diagrams that are useful in modeling reactive objects whose states are triggered by specific events.
- **Activity Diagrams**. Used to model work-flow or business processes and internal op-

[14] Dynamic Link library

eration.

- **Component Diagrams**. Describes the physical software components.
- **Deployment Diagrams**. Describes the physical resources in a system.

To describe the relation between classes in class diagrams, there are some major differences;

Is-a relation. This type of relation illustrates inheritance in a class model. Fig.14 shows this type of relation. In this figure, an inheritance relation between specific types of vehicles and the abstract vehicle class can be seen. This figure reads "a car *is a* vehicle".

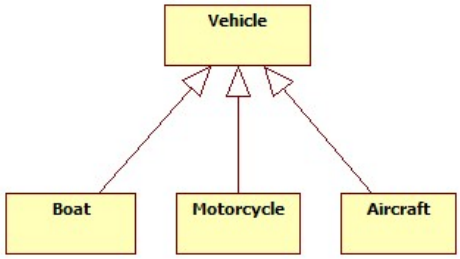

Figure 14: Class inheritance notation

Uses-a relation. This relation, also called *aggregation*, defines a direct relation between two classes. An arrow can be used to show which class is responsible in maintaining the structure. The one responsible is always the class pointing to another class. Multiplicity can be used to show the cardinality of relations. An example of aggregation can be seen in figure 15.

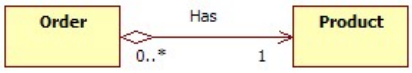

Figure 15: Class aggregation notation

Has-a relation. This type is a *uses-a* relation but it also defines that the classes involved represent a *whole*. If the whole is deleted, all referenced objects[15] will be deleted. If a car is being destroyed, the engine will be destroyed too (fig.16). This type of relation is also called a *composition*.

6.2.3 The Net component

The Net component contains the executive simulation kernel, entities (classes) and relations (associations), the clock, distribution handler and statistics. Fig.17 illustrates how these elements are organized;

The important entities in the model are the agents being simulated.

[15] class instances

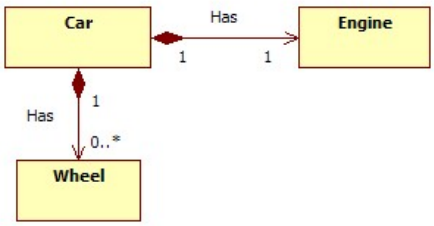

Figure 16: Class composition notation

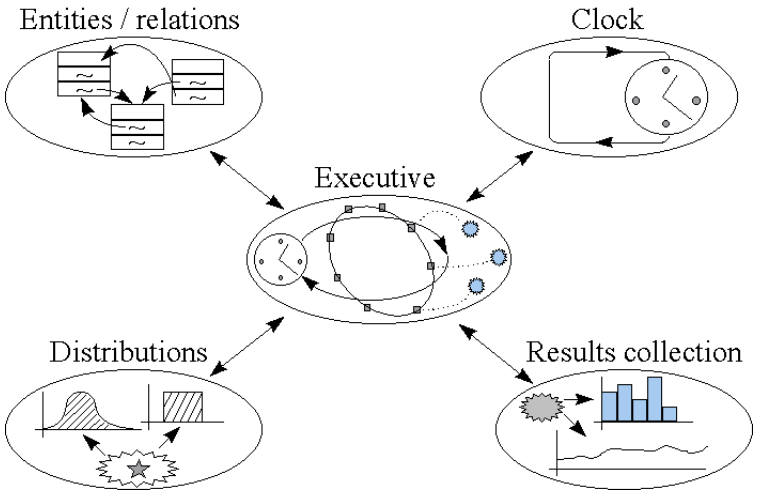

Figure 17: Simulation components

6.2.4 Agent based simulation

Agent based simulation is a type of discrete simulation where agents[16], have their own internal state updated at frequent intervals. A metaphor describing an agent, could be a central processing unit (the agent) in a computer which has its internal state, its registers, updated at sub-sequent clock cycles (intervals). The same applies to most computer games. Agents, like visitors in a park and a train in the commuter system, have states like positions and feelings updated at frequent intervals.

Most computer games uses agent based simulation to simulate the dynamics of the game environment, therefore this will also be the model adopted by the game model in this project.

The main agents will be the network nodes and the virtual users who are using applications to perform network activities. These agents have their own internal states, like active processes, sockets, user feelings, application preferences, etcetera.

6.2.5 The Executive Kernel

The simulation is managed by the executive kernel. The main component of the executive kernel is the simulation loop. Within this loop, the agents are executed once pr. iteration.

[16] simulated entity objects

The loop also manages the timer, distributions of viruses and exploits, and other attacks designed into the model. A decision was made to run the simulation loop on multiple threads to increase the random simulation behavior and also to separate the simulation thread(s) from the main thread intended for managing graphics, forms and dialog boxes. The simulation loop code can be found in appendix D.

Description of the simulation loop in pseudo-code;

1. Exit loop if simulation has been flagged to stop.
2. Update the agent collection if agents has been added or removed since the last iteration.
3. Iterate through, and execute, agents assigned to the loop (thread)
4. If current thread is the main thread (index = 0);
    1. Increase time
    2. If time is a new (simulated) day, accumulate economical statistics and distribute new attacks according to scenario probabilities.
    3. For every 20'th main thread iteration, change score and time statistics
    4. Accumulate trigger statistics
    5. Check end-game trigger flags
6. Sleep a short while and start from the beginning

The static properties of the executive kernel are;

- **Max threads**. The number of threads used to simulate agents.
- **Default timeout**. The default timeout period used by communication protocols. Equals to a pre-defined number of agent executions (kernel iterations).
- **Default ARP timeout**. The number of iterations an ARP entry exists in the resolution manager until it is dropped. A new ARP request will be sent if an entry is missing, and a new entry will be registered with this timeout when a response returns.
- **Ticks pr Day**. The number of kernel iterations that defines a simulated day.
- **Read-mail time**. The number of iterations (time) virtual users spend reading a mail.

The dynamic kernel properties are;

- **Confidentiality importance**. A weighted number indicating how important confidentiality is for a given scenario.
- **Integrity importance**. A weighted number indicating how important integrity is for a given scenario.
- **Availability importance**. A weighted number indicating how important availability is for a given scenario.
- **Initial Money**. The amount of money the player will have when the game starts.
- **Monthly budget**. The amount of money the player receives at the end of each month.
- **Popularity**. A value indicating how popular the player's site is. The frequency of inbound requests from remote sites depends on this property.

- **Maleware frequency**. A value used by the kernel to calculate the probability of new malwares.
- **Local user speed**. The speed at which virtual users at the local site operates.
- **Remote user speed**. The speed at which virtual users at remote sites operates (will also have an influence on the frequency of inbound requests).
- **Spam frequency**. A value used by remote sites to calculate the probability of a spam mail being created and sent to virtual users at the local site.
- **Recruitment frequency**. Used by the kernel to calculate the frequency at which new employees are hired by the company (local site).
- **Retirement frequency**. The probability that employees quit the company.
- **Remote mail frequency**. The probability that virtual users at a remote site creates mails (create-mail tasks).
- **Intranet requests**. The probability that local users want to access a intranet service (in scenarios with a single host connected to the Internet, this value will typically be zero).
- **Initial Malwares**. The number of already existing malwares when the game starts.
- **Auto-create users**. A flag indicating that virtual users (employees) should be created automatically when a new client computer is bought by the player.
- **Description**. A text describing the game objectives.
- **Trigger collection**. A collection of triggers that validates statistical enumerators for end-game and meta-goal situations.

6.2.6 Agents

In the conceptual discussion two types of agents were identified; the network node and the virtual user. These classes were generalized in a base class named *Agent*. The Agent class contains base properties and methods associated with the execution of agents; The enabled/disabled property tells the executive kernel if the agent should be simulated, and it maintains a collection of associated statistics. Fig.18 illustrates the relations between agents and the executive kernel;

6.2.7 Network agents

Network agents are agents that handles network traffic by routing packets using a sub-set of the TCP/IP stack. To simplify the model two distinct types of nodes were identified; Upper-layer and lower-layer agents. Upper-layer agents implements the full OSI stack, while lower-layer nodes contains simple functions for passing packets between upper-layer agents using a sub-set of the ARP and ethernet mechanisms. The *Node* base class maintains common characteristics and functionalities for lower- and upper- layer agents, like;

- **Ports**. A collection of ports and network interface cards used to connect nodes.
- **Packet queues**. Input and output queues/buffers for packets/datagrams.
- **Common properties**. Common properties like a product reference, serial number, name, etcetera.

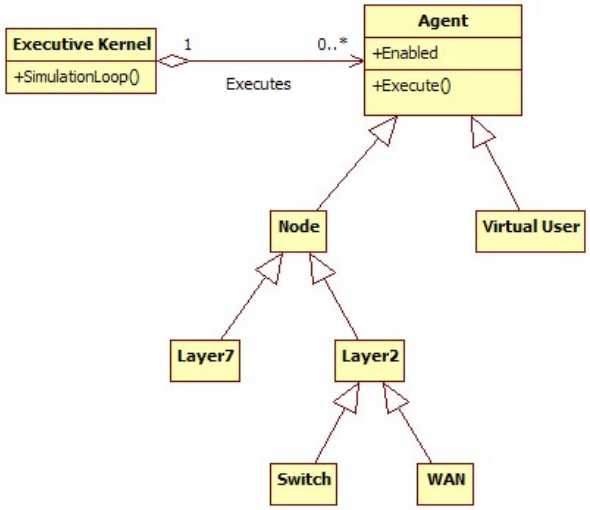

Figure 18: Agents

6.2.8 Virtual Users

"Stupid is what stupid does" -F.Gump.

In addition to net-node agents, virtual users were defined as agents. For simplicity, and to make it easier to distinguish players from virtual users, virtual users will be called Gumps in the following chapters. The motivation behind implementing Gumps is to have some means to simulate network traffic. Gumps have different characteristics, like;

- **Work Efficiency**. How reluctant the Gump is to surf work related web sites and creating useful documents (assets) for the company. This also indicates the Gump's tendency to work with the computer.
- **Equipment efficiency**. The Gump's rate of speed working with the computer.
- **Awareness level**. The Gump's tendency to do secure things, like thinking twice before installing a downloaded program file or visit suspicious web sites.
- **Happiness level**. The happiness level says something about how happy the Gump is. The happiness level will have an impact on what application it wants to use.

Other characteristics was tested, but the one described were found the most useful for the purpose of simulating attacks and user productivity related to Gump behaviors.

The first three characteristics are static values set when the Gump is created. In the future, other parameters could influence these settings. Like when the player, spends budget money on awareness and equipment training, a group of Gumps could have their characteristics improved. This functionality was not implemented in the current version. The happiness level is variable. If the Gump is able to complete tasks it will compliment the system administrator and increase its happiness level. Otherwise, it will complain and get less happy.

Each Gump has it's own state-machine. This is illustrated in fig.19.

State transitions and how long the Gump stays in a given state depends on its char-

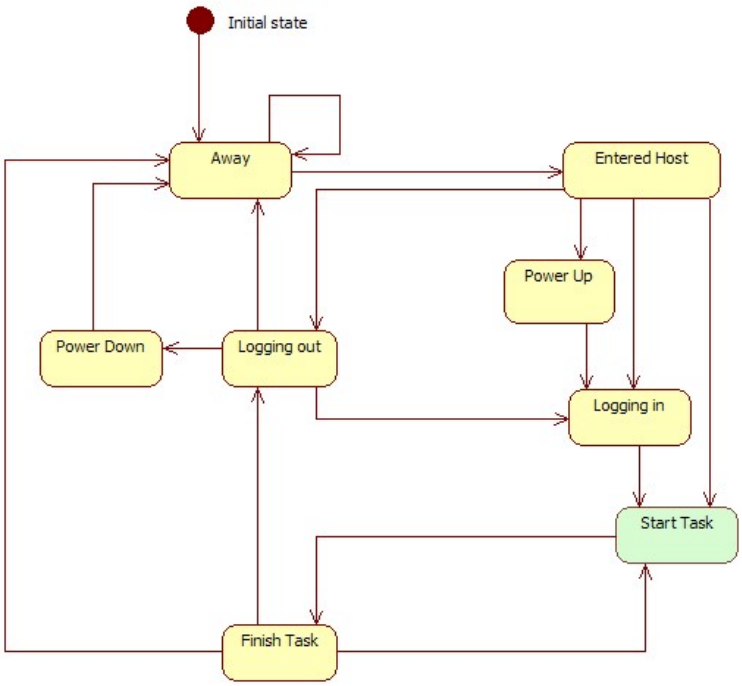

Figure 19: User states

acteristics. When entering the *start task* state, the Gump will try to use an application to create a new task.

The *application* class has the same characteristic types as the Gump; work efficiency, equipment efficiency, awareness and happiness level. In applications, these values are considered characteristics of a typical application user. For example, a file-sharing application could have a lower awareness value than a web browser application. Now follows a pseudo representation of the algorithm used to decide which application the Gump will use to create a task;

1. A Gump mood-characteristic is calculated using a random value and the Gump's characteristics as threshold values. The random value is transformed using a log-function so that mood values, in most cases, will stay close to the "normal" Gump characteristics.

2. Each application known by the executive kernel is iterated and the application with characteristics closest to the Gump's mood characteristics is selected. If more than one application has the same characteristic, a random application among these is selected. A simple distance metric is used to find the closest-match.

3. Then, a method on the selected application called CreateRandomTask is executed (fig.30). This method uses the Gump's characteristics to create a task specific for that application. For a web browser a task could be to surf a web page and parameters for this task could be which web page the Gump want to visit. For Gumps with low awareness levels the random web site selected by this method is more likely to be

"dangerous" than for Gumps with higher awareness. Dangerous sites typically has domain names like nakedpeople.com, warez.com, etcetera.

4. A task is returned with a application specific command, task parameters and a reference to the application and task procedure with the task program code.
5. If the application for some reason fails to create a task, the user will complain and get less happy. The algorithm ends and the user will enter the *finish task* state and wait for a while until attempting to start a new task or leaving the computer. Failing to create tasks could have a large impact on productivity.
6. When a task has been created successfully, all installed products at the host used by the Gump are iterated and products matching the application associated with the task are stored in a collection[17]. A random product is selected to run the task. If no matching product is found, the Gump will complain about a missing application, and the player will need to install a software product for this application on the Gump's computer.
7. If the product already has a process running, the task is handed over to that process and a new thread is initiated to execute the task. If no process is running, a new process will be created first.
8. If the thread fails, the user will complain, otherwise its happiness level will increase.

6.2.9 The Base Node

A base node was designed with a collection of ports. A multiple port design made it possible to have more than one connection from one node to another. Typically, routers could have more than one port, or interface, to connect multiple network segments. A WAN-agent was designed to handle traffic between the local site and the Internet. This node also contains information about the external IP-address range, and the service provider's DNS, POP3 and SMTP server addresses. Multiple WAN nodes could be used simultaneously, thus enabling more than one IP-range per site. Also, interface cards can be associated with more than one IP address. This gives a more realistic model dealing with the configuration of IP-addresses and setting up routing tables, and introduces the ability to use different multi-homing[35] techniques to introduce network redundancy.

6.2.10 Lower-layer Nodes

Lower-layer nodes are nodes that only uses the layers up to, and not including, the network layer. Two different lower-layer nodes were designed; switches and wide area networks (WAN). The abbreviation WAN might be a bit misleading. From a network designer's point of view, the cloud-image in a network map represents a WAN, often a Internet connection. From a site's point of view, this can be seen as an ISP [18] connection, therefore, when the abbreviation WAN is used this can also be interpreted as an ISP connection[19].

A *switch* is responsible for passing packets between upper-layer nodes, nodes that include the network layer and sometimes all layers of the OSI model (hosts). A switch has a number of ports available, and packets can only be sent between nodes if ports are

[17] a host can have more than one product of a specific client application type installed

[18] Internet Service Provider

[19] A WAN cloud can also mean that a network is connected to a remote LAN using other connection routes, but in this model, all communications between different sites goes via the Internet

connected.

When a node sends a packet to the switch, the switch will check the packets broadcast status. If it is an ethernet broadcast packet it will pass the packet on to every connected port. If not, it will pass the packet to the node having its connected network card matching the destination MAC [20] address of the ethernet packet. IP and MAC address mapping is simulated using a simple implementation of the ARP protocol. The use of ARP and MAC addresses will be discussed further in chapter 6.2.13.

An Internet connection is simulated through the use of WAN connections. The WAN connection class sends packets between the different site border routers. Each WAN object has a unique IP address range maintained by a single instance of the Internet class. When packets are sent from a router or host to the WAN, the WAN passes this packet on to the Internet object. The Internet object maintains a table of WAN connections and its IP ranges. The packet's destination IP is then compared with each WAN's address range. When a match is found, the Internet object passes the packet on to the correct WAN which in turn will pass it on to the node connected to the WAN (a router, firewall or host). The Internet is not a agent, its function is to maintain a structure of sites and IP address ranges. Other responsibilities for this object is to create new sites and keep track of, and assign, IP addresses to WANs.

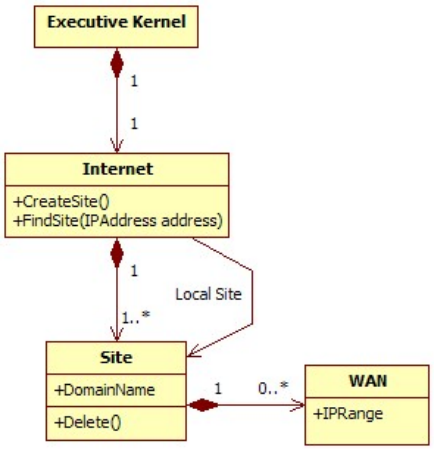

Figure 20: The Executive kernel, the Internet, Sites and WAN

The executive kernel is responsible for creating the Internet object when the simulation starts. The Internet will then create a number of simulated remote sites and a local site that must be maintained by the player.

A decision was made not to include hubs in the model. The reason was the amount of processing needed to broadcast every transmitted packet on to connected upper-layer nodes. A solution to this problem could be to involve a subscription mechanisms for nodes that need to sniff hub-traffic. This solution was not implemented in the current model.

[20] Media Access Control

6.2.11 The Physical Layer

Nodes are connected using ports and interfaces. Nodes can have a variable number of ports installed. Interfaces, or NICs[21], are specialized ports with MAC addresses, while ports are merely "dumb" connection points. Interfaces are typically used by upper-layer nodes to connect to other nodes, thus involving the ARP mechanism.

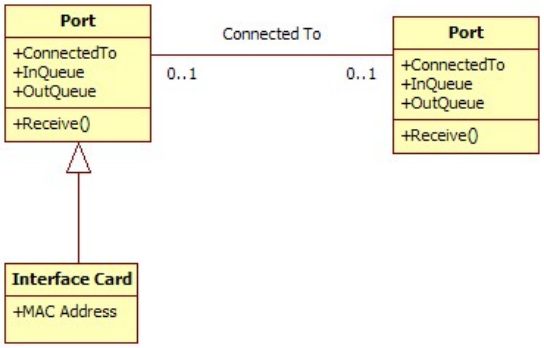

Figure 21: Ports and Network Interface Cards

6.2.12 Upper-layer nodes

These are the nodes that implements the network layer and maintain a complex state structure. Typical upper-layer nodes are; client computers, servers, routers and firewalls. The state structure for upper-layer nodes is shown in fig.22.

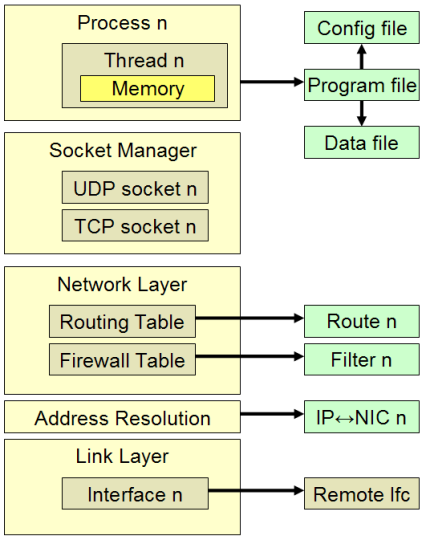

Figure 22: Host model

By enabling and disabling layers in this structure, it was possible to create;

[21]Network Interface Cards

- **Routers**. Operates at the link and network layers. Contains a route manager/table with a variable number of route definitions. Routers have layer-entities above the network layer disabled. Routers does not activate the firewall mechanism.
- **Firewalls**. An advanced router with the firewall mechanism activated.
- **Clients and Servers**. Activates all layers but not the firewall mechanism and operates as physical end points in network communications.
- **Clients and Servers with firewall**. Activates all layers including the firewall mechanism.

6.2.13 The Network layer

In the network layer, the routing and address resolution mechanism can be found.

Properties of the routing table;

- **Net**. The network address.
- **Gateway**. The gateway IP address. The net address belongs to the local host if the gateway is 127.0.0.1 (loopback address). Otherwise, if another IP is used, this would be the forward IP for IP packets matching the net address. If not set, the net address belongs to the local sub-net and the packets destination IP and ARP are used to address the destination node.
- **Port**. The local port associated with the net address
- **Metric**. This was not implemented. Reserved for future use to decide which route should be selected when multiple routes are available.
- **Route protocol**. Dynamic routing was not implemented.
- **Flags**. Not implemented. Reserved.

Sub-net table entries only use the net and gateway properties. The gateway property is always the IP address of the router en-route to the given net address. If the gateway IP is not set, the net address should be considered being a local sub-net IP range.

The address resolution manager was included to make it easier to handle changes in IP addresses and connections during the simulation. If a connected node changes its IP address(es), or disconnects, communication sessions will time out until the IP and MAC address mapping times out and gets removed from the mapping table. When a packet is about to be forwarded to another IP address and the IP address is not found in the address resolution mapping table, an ARP request will be broadcasted on connected network segments. When a node receives an ARP request and the queried IP address is found at the receiving node, a ARP response will be sent back to the requesting node. Finally, a new address map will be registered with a new defined time out value. When the ARP mechanism send an ARP request, the packet waiting to be forwarded is placed in a pending table. If this request times out, the pending packet will be marked as unreachable and dropped. This sub-set functionality of the ARP was found sufficient for the simulation. Other ARP and RARP mechanisms was not implemented.

The firewall maintains two collections of filters organized in chains. Filters contains rules and can be added, deleted or reorganized by moving them up and down the filter chain. The main purpose of this chain is to allow or deny an IP packet to pass. When a packet from the link layer is received at the network layer, it will be checked against

the firewall's inbound rule chain. Packets sent from the network layer to the link layer will be checked against the outbound rule chain. The default rule can be used if no match is found in the filter chain. The default rule can be to deny or allow the packet in these cases. Separate rules can be set for inbound and outbound filters. Fig.23 shows the classes involved in the network layer.

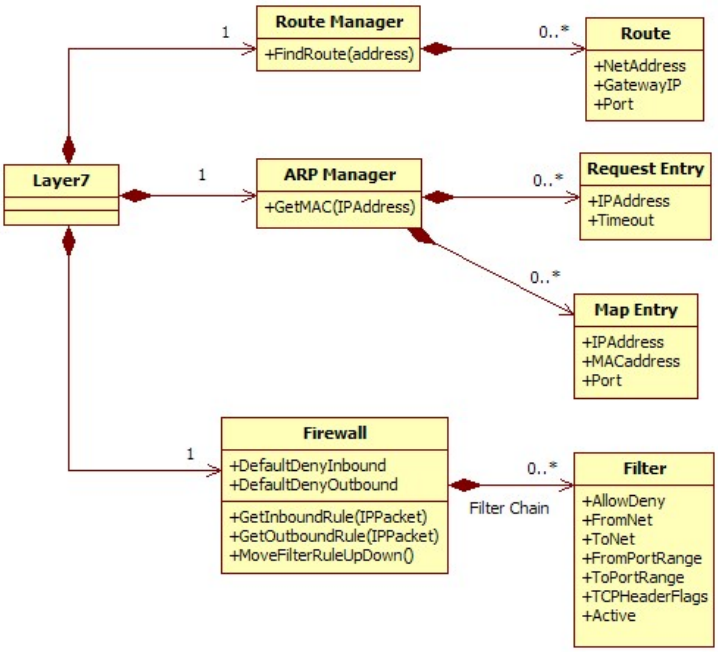

Figure 23: Layer 3 classes

When a packet arrives at the host and the best matching route has a gateway IP equal to 127.0.0.1[22], the packet has reached the receiving end-point and is handed over to the transport layer.

6.2.14 The Transport Layer

The transport layer is enabled in host-nodes. Two types of host nodes was defined; clients and servers. The state model for these are the same, but they are threated differently in the way they are managed. Servers are handled by the player only, while client computers are managed by Gumps and the player.

The transport layer manages the state of connections and maintains a collection of sockets. Sockets are logical units used by threads in application processes to receive and send data packets.

If a packet using the TCP protocol is received from the network layer, and the SYN flag is set, a TCP socket in listen state listening on the same port as the packet's destination port will be used as the destination socket. Otherwise, the transport layer will try to match the address/port-pair of the packet with any of the active sockets using the same protocol. If a matching socket is found, the packet will be passed on to the socket's input

[22]Also known as the *localhost* or *loopback interface*

queue, otherwise the packet will be dropped.

When a process is sending a new data packet using a socket, this packet will be sent to the transport layer, wrapped with the corresponding protocol header, placed in the output-queue and passed over to the network layer to be routed to its destination.

6.2.15 Data

Data can be *anything*. It was decided to define an abstract data class to include common properties like size and a description of data. Two main classes was defined to contain data; packets and files. In addition a packet data class was defined, a type of *data* class defined as a structure that could be used by applications to transmit application protocol data, including a command string, data related to the command and a reference to an exploit (packet-exploit), if any.

6.2.16 Packets

To simulate the TCP/IP stack, different types of packets was defined, each representing specific layers. At the application layer, processes can send and receive *data*, files and packets, using send and receive operations on sockets. The send sequence is;

1. The process creates a data packet containing the protocol command and associated data, if any, and
2. uses the socket's write operation to send the data packet.
3. A transport packet is created in the transport layer using the protocol defined by the socket (UDP or TCP). This packet contains the information about the source and destination ports used by the communicating hosts. The data packet becomes the payload of the transport packet. Typically, ephermal ports are used by client sockets.
4. In the network layer, an IP packet is created and the transport packet becomes the payload of this IP packet.
5. When the routing mechanism in the network layer has found the forward IP, it's MAC address and the physical port to transmit the packet on to, an ethernet packet is created using the IP packet as payload.
6. The ethernet packet is sent to the physical port and onto the connected unit, typically the switch, which in turn passes it on to the connected node with the interface card matching the destination MAC address of the ethernet packet.

The sequence when receiving a packet;

1. The ethernet packet is received at the physical layer (interface card)
2. The payload (IP packet) of the ethernet packet is passed on to the network layer along with information on which port it was received on.
3. If the destination IP matches the node's IP address, the payload of the IP packet (the transport packet[23]) is passed on to the transport layer.
4. The transport layer maintains a collection of active sockets, and locates the socket using port and protocol information from the transport packet. The data packet (innermost packet) is passed on to the socket's input-queue.

[23] TCP or UDP packet

5. The process receives the first data packet in the input-queue at the next socket read-operation.

The onion-design, wrapping different protocol packets as payloads, made it easier to duplicate packets with the same inner payload. This comes in handy when packets are broadcasted and handled by multiple stacks simultaneously. Another design, using one class with properties from all layers in one packet was tested, but the idea was soon discarded due to lack of flexibility.

Fig.24 shows the inheritance relation between data, packets, data packets and files, and the packet onion-design.

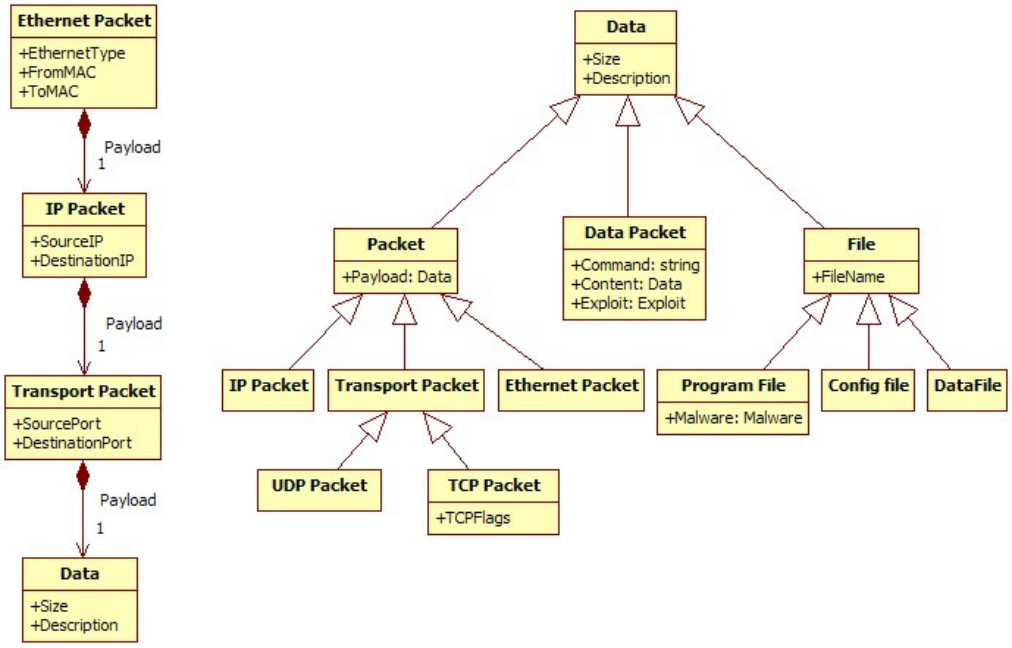

Figure 24: The Packet Structure

6.2.17 Data flow

Agents create data packets using applications and processes. Processes uses sockets to transmit data packet on to the communication stack. Data packets are not agents, but objects used, and transmitted, by agents. Packets are sent from node to node using queues. Figure 25 illustrates this concept.

6.2.18 The File System

An important part of the host-state model is the use of files as non-volatile state objects. This means that parts of the host state is restored when a host is powered on after being shut down. These states are the reason why viruses and other malware resides, and a simple file system had to be implemented.

Four file types was defined;

- **Program files**. Defines a relation with a software product (ref. chapter 6.2.20), the

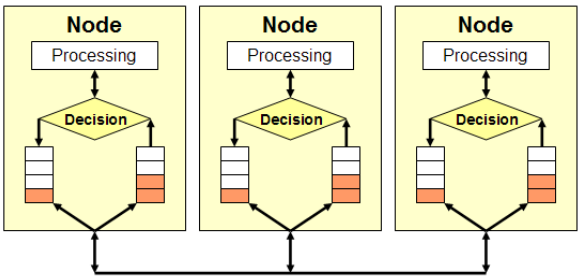

Figure 25: Traffic Flow

latest patch, a reference to a malware object if the program file is infected, and a reference to its configuration file.

- **Config files**. Configuration files contains non-volatile configuration properties used by processes created using the associated program file.
- **Data files**. These are files containing data. Data could be a spreadsheet, a word document, a drawing, a database, etcetera. Some applications creates or uses data files (assets).
- **Folders**. Folder files maintains a collection of files (or folders), creating the hierarchy seen in real-life file systems. Typically, when a program file is installed, a new folder will be created containing the program file and configuration file.

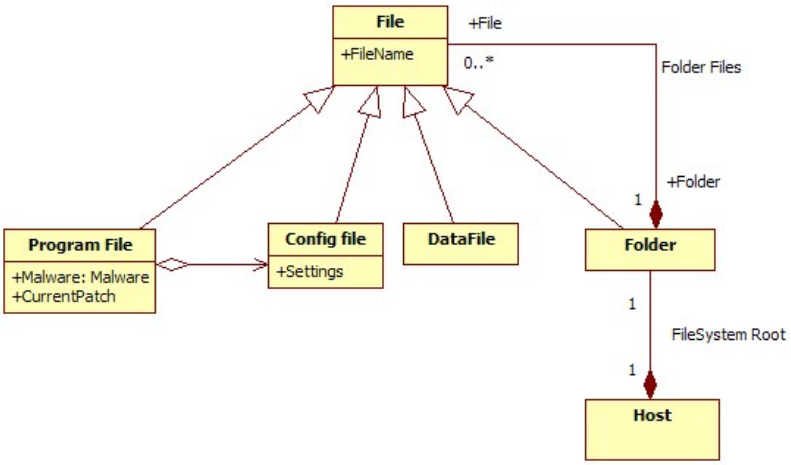

Figure 26: Files and the File System

Some folders are created automatically when a new host is created. These are;

- **Downloads**. Gumps uses different applications to download data and program files. When program files have been downloaded it will be placed in this folder. Later, Gumps might install programs found in this folder.
- **Documents**. This is where documents (data files) downloaded or created by Gumps

are stored.

- **Installed**. A folder containing a list of soft-links to installed program files.
- **Services**. Contains a list of soft-links to installed server-application program files

6.2.19 Sockets

Sockets are the logical end-points used by processes to communicate with other hosts (or processes) using the TCP/IP stack. It was decided to implement both the UDP[24] and TCP[25] protocol. When a thread creates a socket it must decide which one of these two protocols it should use to communicate with a remote process. There is no reason to go into detail on how these protocols operates. Excellent resources are [34], [39] and [36].

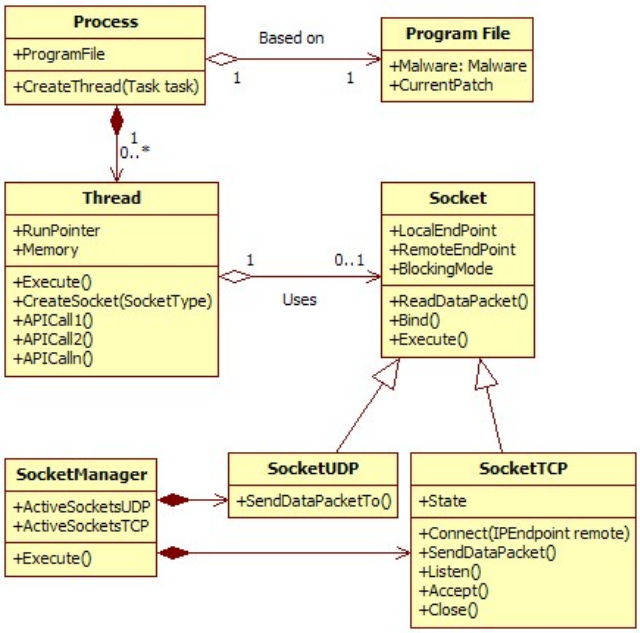

Figure 27: Threads and Sockets

IANA[26] is the responsible authority for managing UDP and TCP ports in the range 0-1023. In addition they provide a list of registered ports in the range of 1024-49151 for the convenience of the community. The full list can be found on web page

`http://www.iana.org/assignments/port-numbers`.

Other ports can be used at the user's discretion, but using well-known ports defined by IANA is recommended.

To simulate basic TCP functionality, the full state diagram for TCP was implemented. This state model is shown in fig.28, an illustration borrowed from [34, p 241] and [36].

Processes should now be able to act as both servers and clients. Server processes typically creates TCP sockets and starts listening on a standard, or registered, port specific for the application.

[24] User Datagram Protocol
[25] Transmission Control Protocol
[26] Internet Assigned Numbers Authority

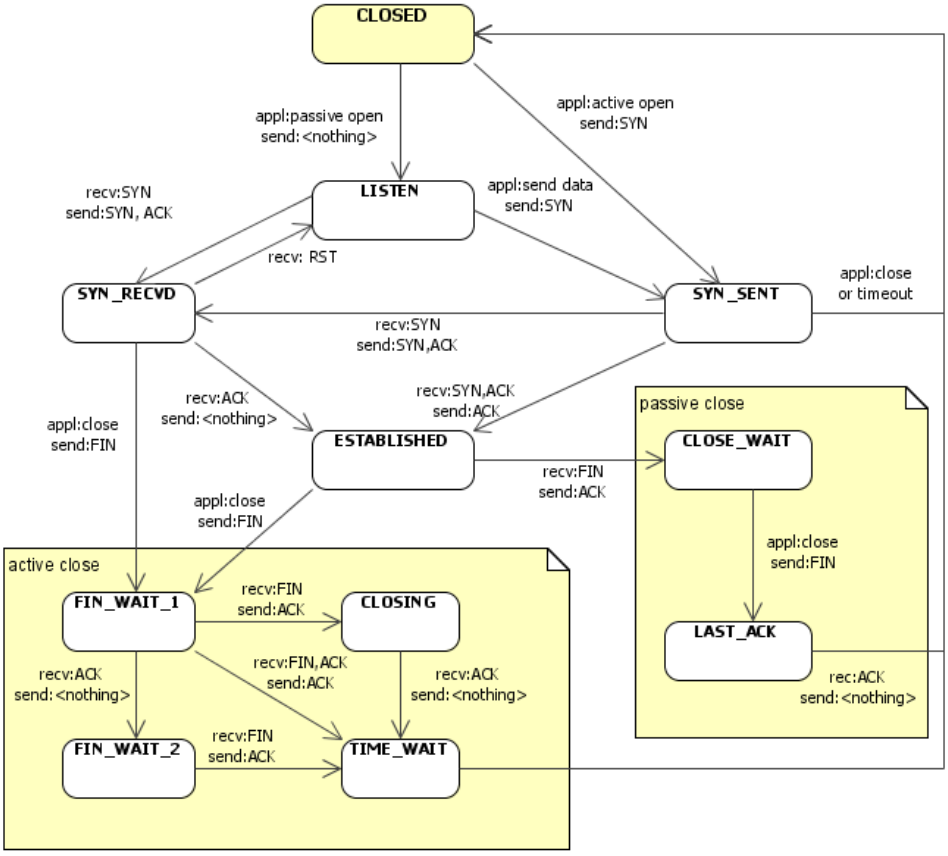

Figure 28: The TCP state model

A simple version of the TCP transmission functionality was implemented. The implementation did not include;

- IP fragmentation,
- the use of TCP window size, and;
- sequence and acknowledge numbers.

Data is transmitted in one packet regardless of the data size. A typical session would be like this;

1. Connect to remote host using the handshake mechanism.
2. Send and receive data using one data packet to transmit the complete data packet and wait for response. This sequence can be repeated as many times as the application protocol requires.
3. Tear down the connection, either by letting the server close the connection, or more commonly, letting the client close it.

The UDP protocol is connection-less. Processes must send the destination IP and port along with the data packet when sending data to a remote host.

6.2.20 Running applications

Host agents are able to run a variety of applications. Three different classes were defined to run applications;

- **Application**. This base class defines the algorithms and characteristics of specific types of applications, the name of the application type[27], and other characteristics associated with the application type.
- **Software Product**. Defines a specific software application, the product name[28], performance values for characteristics defined by the associated application, and other properties like version and vendor.
- **Program file**. A program file is an host-instance of a software product. These are the objects that can be infected by malware. Program files are installed into the host's file system.

Fig.29 shows the relations between application, software and program-file classes.

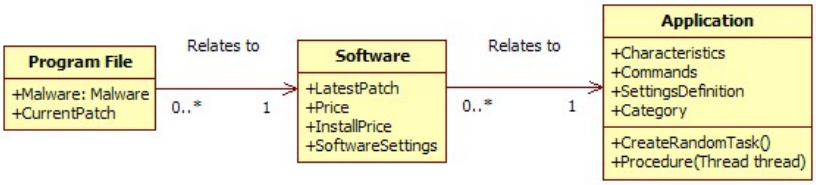

Figure 29: Applications

6.2.21 Processes and Threads

Processes and threads are part of the host's volatile state structure. Processes are automatically created for installed server applications (services) when a host is powered on. Client application processes are created by Gumps when they want to execute a task. Gumps tend to shut down processes when they have been idle for a while.

When a program file has been installed in the file-system, a reference is put in the *installed* folder. When a virtual user need to start an application, a program file associated with a software product and the application in question are used to create a new process. If there are more than one software product, a random program file (product) is used. This could be the case if, for instance, more than one web browser is installed. When the process starts for the first time, a loading task is initiated, creating a thread for the initialization code. Any pending tasks are queued until this task ends. Then a new thread is created for every queued task. Sub-sequent tasks requested by Gumps are converted to threads and executed by the process.

With these specifications, the model should be able to simulate advanced process and thread handling. Every time a host-agent is executed by the executive kernel, each process and its threads will be executed, moving the thread's running-pointer forward one instruction at the time.

The thread procedure is located in the application class. This way, all software prod-

[27] Web server, Web browser, Mail Client, etcetera
[28] Internet Explorer, Opera, FireFox, etcetera

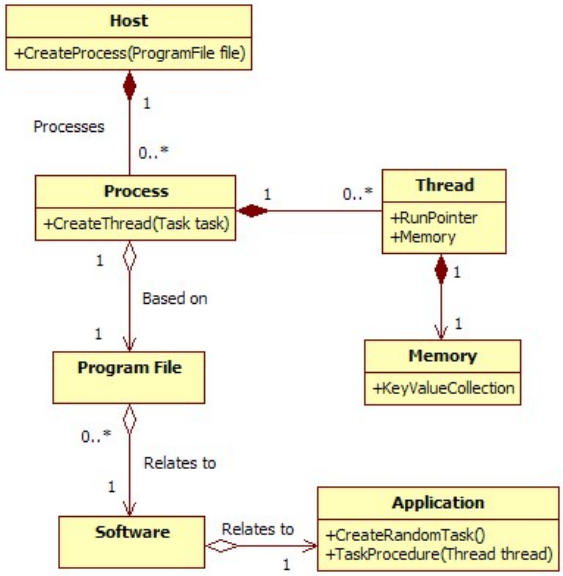

Figure 30: Processes and Threads

ucts associated with the same application uses the same procedure. Even so, products can have different capabilities as each product have their own distinct performance characteristics. For instance, a WebServer application defines the "MaxConnections" capability. Different web server products can have their own values for max connections. Typically, the more expensive a product is, the better it will perform. A sample of such a procedure can be found in appendix E.

6.2.22 Memory

Threads can read and write data to a *Memory* class. Memory is a part of host's volatile state structure. This class maintains a collection of key-value pairs. This mechanism is useful when a thread needs to *remember* a value between consecutive thread-executions.

6.2.23 Malware and Exploits

Virus applications and exploits are created by the malware factory class. The Internet holds a reference to this factory and malwares and exploits are created at random intervals with a frequency defined by the executive kernel. The executive kernel calls the malware factory's CreateRandomMalware method which in turn creates a random malware task with random behavior parameters. Parameters can be how many files a malware task should try to delete or send pr. e-mail to a remote receiver. Only a few malware tasks were built for the game prototype;

- **Delete Random Files**. Deletes a random number of files from the infected host. Files are marked as deleted using a shadow-deleted flag on the file. If the Gump tries to open a shadow-deleted file, it will complain that the file is missing.
- **Send documents to any**. A random number of documents (assets) are sent to random users at remote sites.

- **Crash a process**. Infected processes crashes at random intervals with a random probability.
- **Infinite Loop**. Takes a task as a parameter and runs this in a new thread at random intervals.

It is possible for a thread to spawn new threads. A thread can also wait until another thread has finished. This makes it possible to build complex sets of malware structures.

The attack diversity could have been much higher if there had enough time to implement other more sophisticated malwares, but the current design should make it relatively easy to extend this set in future projects. The application-process model should make it possible to create malwares that use sockets to connect to and infect other services on internal servers. Also, malwares acting as servers (trojans) and malwares that downloads malicious programs from remote sites, could be implemented. Having these types of malware it would be more important for players to use design principles[6] [7] to prevent malwares from spreading between hosts on the internal network.

When a new malware or exploit has been created, three (or four) different random milestones are registered;

- **Created Date**. The simulated day the new malware is created.
- **Detection Date**. The day the malware is detected.
- **Signature Date**. The day a signature will be available for typical, average, anti-virus products (delta signature day=0).
- **Patch Date**. The day when a patch will be available for the product targeted by the exploit.

When an infected data file is opened or an infected program file is started, the associated malware will automatically spawn a new thread running the malicious code in the callers process. If the malware is set to infect running processes, active processes will also be marked as infected with the malware and new malware threads will be spawned for these processes when they restart.

6.2.24 Script-kid attacks

A script kid application was created. This application is installed on servers at randomly selected remote sites, typically sites with a low trust-level. At random intervals these application instances tries to connect to a random non-public services at the local site.

Port-scanning scripts was not implemented, instead the script-kid tool uses model-data to find, and target intranet servers. If intrusion detection equipment and software, other than the implemented anti-virus products, are implemented in the future, port-scanning scripts could also be implemented to extend the attack simulation.

6.2.25 Spam

Mail server and mail client applications were implemented to handle mail traffic. When Gumps at remote sites send e-mails, a random number of these are marked as spam. When a local Gump receives mail, it will spend some time reading it, time that could have been used to create assets. Hence, spam reduces the overall Gump productivity.

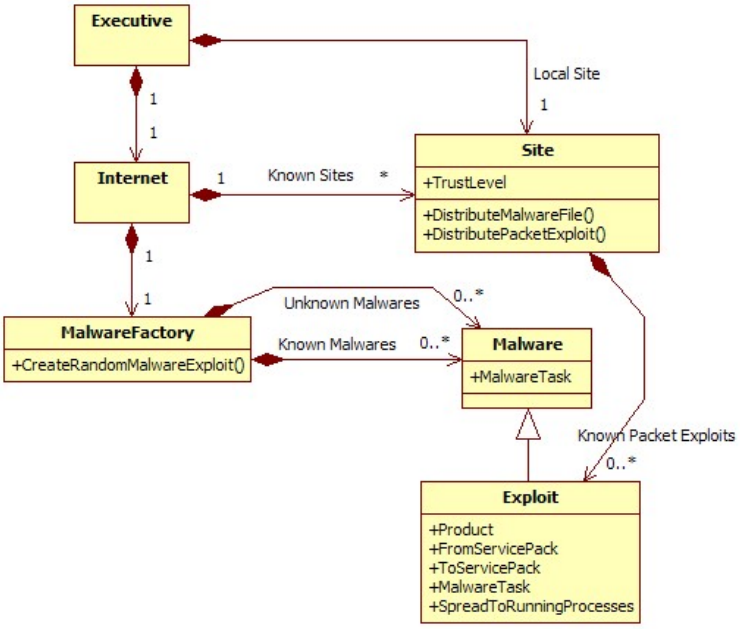

Figure 31: Malware and Exploit distribution

6.2.26 Countermeasures

The first line of defense is how the player designs his/her network. The player should use design principles found in [8] and [6] to prepare for attacks. In the prototype this is not as important as it should have been due to the low number of malware tasks implemented, but still, script-kid events could be stopped by using firewalls correctly.

From the time a malware is created and until it is detected, the player must rely on the design of the network. When a malware is detected, a news-flash occur which describes what the malware intends to do (see fig. 32.

Knowing what the malware does, the player could use host-hardening techniques, like shutting down related services or hosts to tighten security.

An anti-virus application was defined together with a set of products having different capabilities. Each product has its own distinct delta-signature value. This value is used along with the maleware's signature day property to decided when new viruses are detected. Anti-virus products with a low, negative, delta value will detect viruses earlier than cheaper products with a higher delta value. Another capability is the false rejection rate. Cheaper anti-virus products will more often raise false virus alerts.

Scenarios can be defined to have their own POP3 and SMTP services. If not, the ISP associated with the Internet connection provides these services. By design, an ISP does not offer spam-filter services, so players should consider to install their own mail servers and spam-filters. Spam-filter capabilities are related to false rejection rates (FRR) and false accept rates (FAR). Cheaper spam-filter products with high false acceptance and rejection rates will more often accept spam mails and reject good mails.

Finally, for exploits, a patch is released some time after the virus signature has been updated. The player could then patch installed software to remove any vulnerabilities

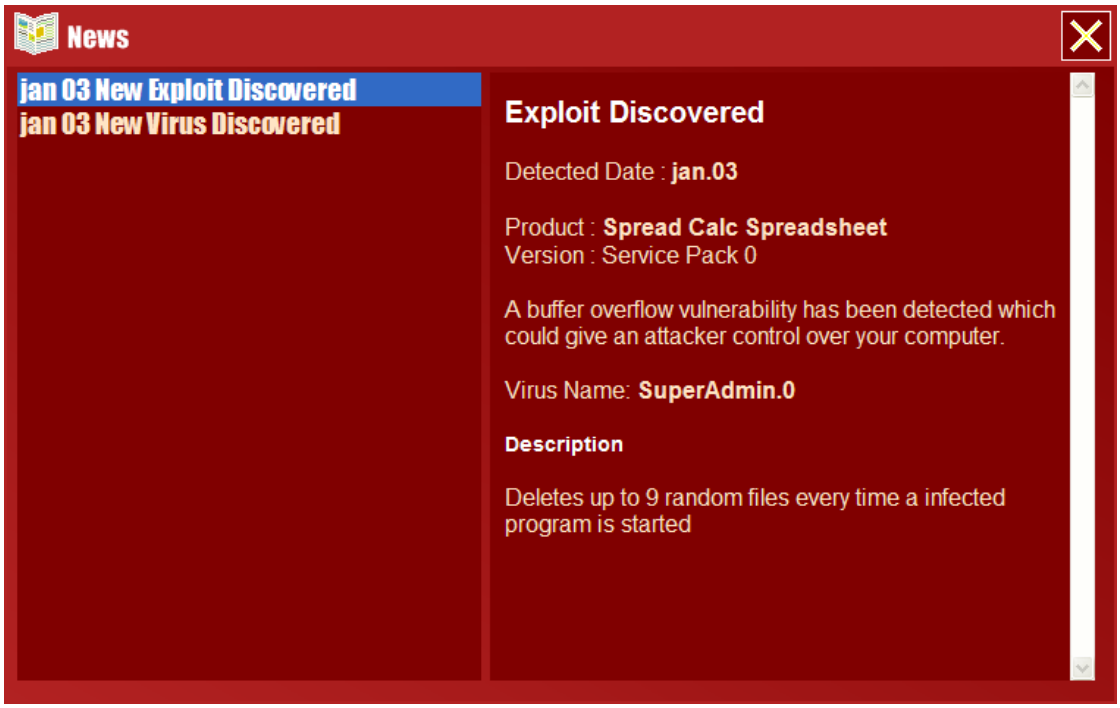

Figure 32: News flash

associated with an exploit. But, patching comes at a cost. The player should have in mind that new exploits could introduce vulnerabilities targeted by future exploits. Exploits could target specific ranges of service packs, not only up to a given service pack.

The player should carefully consider to use different products, or the same product, for any given application. Using different products could reduce the overall damage of an infection, but might cause host to be infected more often. Using the same product will possibly cause less frequent infections but might cause greater damage.

Finally, the player can use a remote file server to store documents and other files created or downloaded by the Gumps. This introduces a single point of failure so these servers needs to be secured using firewalls, host-hardening techniques and anti-virus products. By installing a backup-tool the player will gain more points because files are stored on a secure medium. If the player chooses not to use a central backup facility, documents will be stored at client hosts. When reaching a certain amount of files, the user will attempt to extract files and store it on less secure medium. Cheaper backup tools will back up files less frequently.

6.3 Summary

In this chapter the process of building the game model and prototype has been described. To evaluate the results from this research an experiment was conducted. This experiment is described in the next chapter.

7 The experiment

To investigate to what extent the game satisfied the theoretical characteristics for being a fun game, a user trial was initiated and a feedback questionnaire was constructed. Finally, telephone interviews and personal interviews was arranged.

7.1 User Trials

After five months of development, a setup kit was constructed and a user trial was arranged. The setup kit was made available at the project's web site[1]. In addition to measure how the experienced enjoyment was, knowledge about the game's face and content validity was needed; if the game looked like a network and host security game and if it included elements associated with this topic. Due to the complexity of the game, a set of candidates with prior experience in network administration and information security was identified and contacted.

Participants was asked to download and play the game. The trial period was variable in length, some participants were contacted immediately after the setup kit was made available and these were able to play the game for a couple of weeks. Some were able to play for a few days. There were no restrictions on how much (or little) they played. During the trail-period, the researcher's role was to help the participants with technical issues.

A FAQ[2] section was created at the project's web site and participants had access to an on-line user manual during the trail period. The user manual can be found in appendix B.

7.2 The Feedback Questionnaire

A feedback questionnaire was made available along with the setup kit so that participants could prepare for the interview. The feedback questionnaire can be found in appendix A. Ordinal questions was constructed in relation to Malone's intuitions, categories from the GameFlow theory, and topics related to the face and content validity of the game. Finally, questions were added to catch open-ended answers related to the game experience.

7.3 The Interviews

During the interview the researcher's role was to clarify any questions participants had regarding the statements and questions found in the questionnaire. Interviews were conducted as a structured interview, and questions were asked in sequential order as defined by the questionnaire. To reduce the bias in answers, participants were explicitly told to be honest in their responses. Both telephone interviews and personal interviews was arranged.

[1] http://www.stud.hig.no/(tilde)040673
[2] Frequently Asked Questions

8 Results

In this chapter, results from the interviews are presented.

8.1 The Participants

Six candidates agreed to participate in the experiment. Demographic data and prior knowledge in network administration and network security for those who participated were;

- **Subject 1 [S1]**. Male, Age:34, MSc, Network: 10 years, Security: 8 years
- **Subject 2 [S2]**. Male, Age:34, MSc, Network: 15 years, Security: 5 years
- **Subject 3 [S3]**. Male, Age:24, MSc, Network: 2 years, Security: 2 years
- **Subject 4 [S4]**. Male, Age:24, MSc, Network: 2 years, Security: 2 years
- **Subject 5 [S5]**. Male, Age:29, MSc, Network: 1 years, Security: 2 years
- **Subject 6 [S6]**. Male, Age:35, Bachelor, Network: 2 years, Security: 1 year

8.2 Ordinal questions

In fig.33, quantitative values for the ordinal statements are presented (ref. appendix A). This figure also includes the actual response values. Response values represents;

- 1 = Not at all
- 2 = No
- 3 = Maybe (neutral)
- 4 = Yes
- 5 = Absolutely
- (blank) = Don't know

The *mode*, *median* and *average* values for ordinal responses can be found in figure 34.

8.3 Open-ended questions

The participants were presented with seven open-ended questions. Responses to these questions are described in the following chapters. [Blank] indicates that the participant failed to respond.

8.3.1 The negative properties of the game?

- **S1:** The phase or speed was too fast. Events like new viruses and exploits occurred to frequently. The Graphical user interface could be improved even more.
- **S2:** I was able to play, but experienced a crash. The speed should be adjustable.
- **S3:** A little too much information for beginners.
- **S4:** Much information. Steep learning-curve.

- **S5:** To much information. Should be able to hide the network-traffic sniffer window. Missing help-functionality regarding router and firewall configuration. Too many events occurred and too frequently. Simulated dates should change at a slower phase. Maybe the game should be paused during configuration tasks.
- **S6:** The game has a pretty steep learning curve. It result might be that some players give up before they have really got started at all.

8.3.2 The most negative property of the game?

- **S1:** The game speed.
- **S2:** The game crashed.
- **S3:** A little too much information.
- **S4:** To much happening at the same time. It should be possible to delay the simulation.
- **S5:** To much information.
- **S6:** [blank]

8.3.3 The positive properties of the game?

- **S1:** The graphical user interface. The game concept.
- **S2:** The user interface. When you connected nodes things started to work.
- **S3:** Funny and fascinating. A good way to learn employees about data security.
- **S4:** Many fundamental network elements included and that it is close to the real thing.
- **S5:** The user interface. Virtual users are close to the real thing.
- **S6:** The graphical user interface awesome. It is also positive that the game is loaded with humor.

8.3.4 The most positive property of the game?

- **S1:** The game concept.
- **S2:** The interface. Icons looked appropriate
- **S3:** The graphical interface.
- **S4:** It felt real. Figures and icons were recognizable.
- **S5:** It felt real.
- **S6:** The graphical user interface.

8.3.5 Ideas to make the game more fun and interesting?

- **S1:** Create an on-line multi-player version. Include a game-level editor so that players can create their own scenarios. Include physical security. Make it possible for players to create their own services, applications and communication protocols. It should be

possible to construct hardware components (network nodes) too.

- **S2:** A lot of fancy and funny graphics. Visualizing attacks.
- **S3:** Include a multi-player option. Reduce the amount of information, especially the sniffing-information. More hardware types like IDS and VPN components.
- **S4:** More hardware and software elements. Background music. One should be able to view the network at different abstractions using a zoom-function.
- **S5:** Multi-player option where new exploits could be downloaded from a game server and be used to attack other players. The game should include music and sound effects.
- **S6:** It would be great to be able to choose side (security responsible or hacker), and then be able to play this as a multi-player game via the Internet.

8.3.6 Ideas to make the game more motivational?

- **S1:** [blank]
- **S2:** [blank]
- **S3:** [blank]
- **S4:** By including a multi-player option in a learning-environment. Players should be able to construct their own scenarios.
- **S5:** [blank]
- **S6:** There should be some kind of bonuses when you achieve your goals. Either a nice picture, a short movie, or maybe a cinema ticket discount.

8.3.7 Other comments

- **S1:** [blank]
- **S2:** [blank]
- **S3:** [blank]
- **S4:** [blank]
- **S5:** [blank]
- **S6:** [blank]

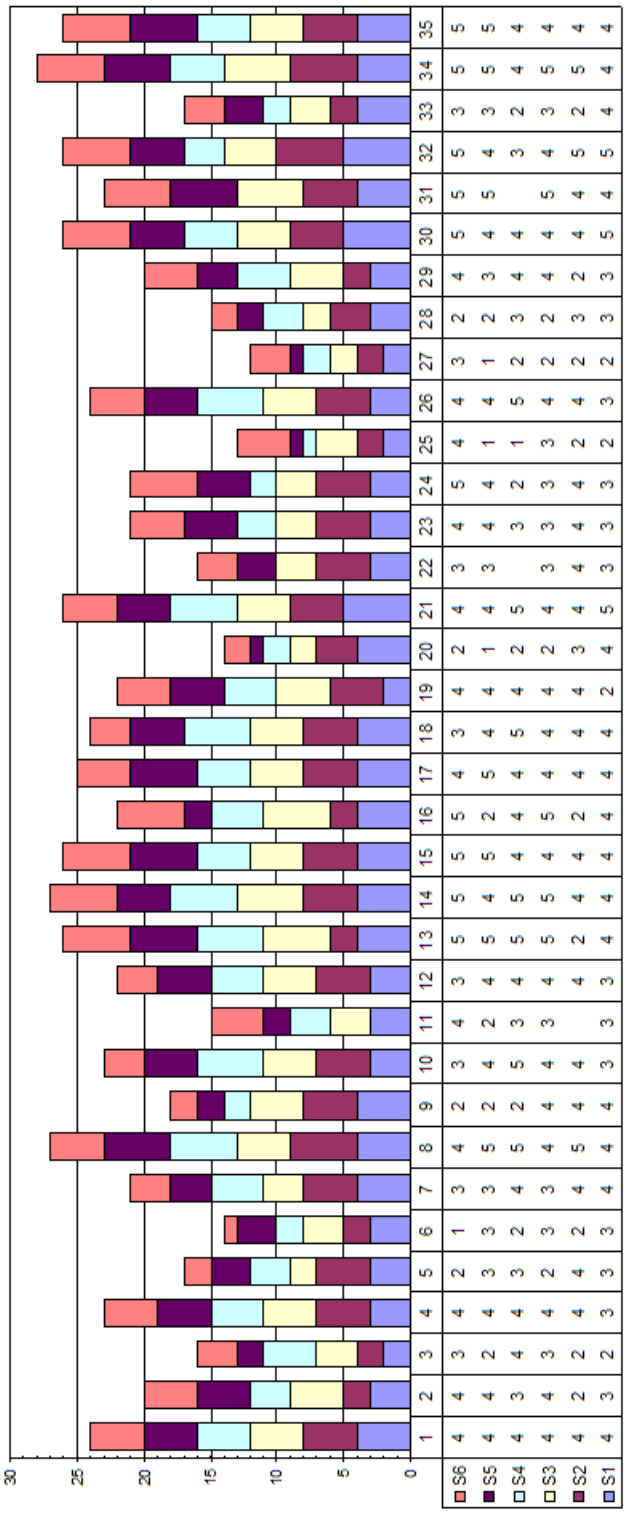

Figure 33: Results

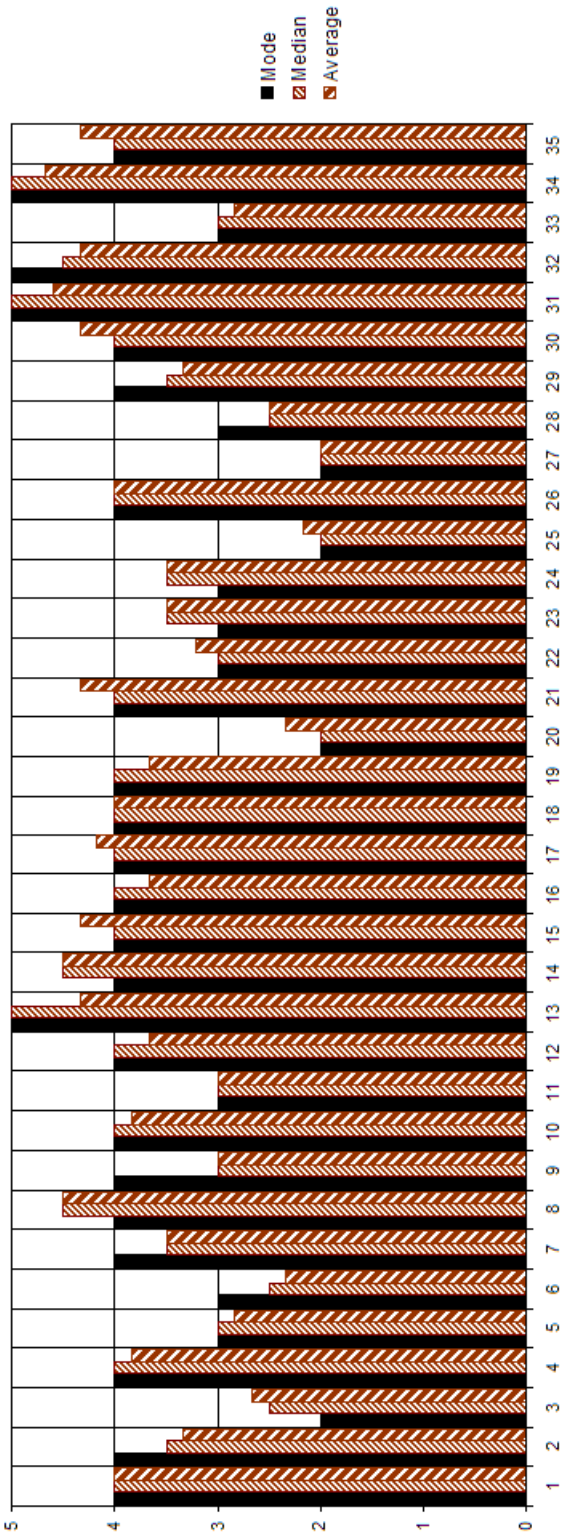

Figure 34: Mode, Median and Average

9 Discussion

In this chapter results from the design research and results collected from the user trails and feedback interviews will be discussed.

To test the hypothesis stating that the game prototype should be fun to play, results will be discussed using Malone's[14] intuitions and the categories found in GameFlow[15]. Each category receives a grade in accordance to the results received in the experiment. Finally, a conclusion will be made from these grades.

But first a short discussion on experiences from the design research phase.

9.1 Design and development

The design research method was found suitable in building the game model and prototype where model ideas was accepted and rejected in an iterative process using theory and engineering skills.

A total of 1270 man-hours was spent on developing the game model and prototype within a period of five months.

9.1.1 Model Issues

Developing a model for simulating networks and hosts with complex state structures proved to be hard and time-consuming. Theory from the literature had to be selected carefully as the work progressed and the model was extended. Considerations had to be made in balancing the model between simplicity and the need for a structure that was complex enough to simulate attacks. This structure showed that a wide range of malicious tasks could be constructed. A small number of malwares and exploits were introduced to create attacks, but even more of these could have been constructed to add more challenge to the game.

9.1.2 Sensory Issues

The decision not to use accelerated graphic technologies, made it easier and faster to implement the graphical interface, but accelerated graphics could have improved the animation-rate and graphical effects, which are crucial elements in enhancing the game fantasy, and should therefore be considered in future versions.

The most difficult and time-consuming item to implement, related to the graphical interface, was the interactive network map. This part required some skills in creating graphical editors, and artistic and creative skills in designing the graphical symbols. To achieve better results, graphical artists should be involved in this part of the process.

Music and sound effects were not implemented. Sound is an important element in games, but require extra resources with knowledge on how to create music and sound effects and how to adapt this in the best way to enhance the fantasy.

9.2 Feedback results

The following chapters discusses the results from the questionnaires in the context of Malone's intuitions[14] and the characteristics of GameFlow[15]. Using user-trails and

feedback questionnaires seemed appropriate for this research. Only six volunteers participated in the interviews, which is not sufficient enough to draw any statistical conclusions. Instead results will be discussed qualitatively which could give an indication of the levels of experienced enjoyment. The second intention was to measure the face and content validity of the game in relation to network security. Therefore, using participants with prior knowledge seemed to be appropriate. This was not a part of the research question, but relevant to show that the game indeed was valid in a network security context.

The open-ended questions about ideas to make the game more motivational and the question requesting other comments was a bit confusing and should have been left out.

9.2.1 Malone: Identifying with the Fantasy

The game was designed so that players should act as network administrators and security managers. In question 8, participants were asked if they identified with this role when they were playing. Answers were equally distributed between *yes* and *absolutely*.

Responses were almost identical in question 30, where we asked if the concept was good and well suited for network security.

This indicates that participants experienced the game as valid for network administration and security management, and thus confirming the game's face and content validity. Results also indicates that players with prior knowledge in this area could adapt to the game fantasy quite easily. Due to the participants' background, we can not tell if players with no prior knowledge would adapt to the fantasy in the same way. This could be a topic for future studies.

Does fantasy contribute to enjoyment? Yes.

9.2.2 Malone: Sensory

The game did not include music and sound effects, but questions related to the game's graphical representation was asked (question 13 to 16).

Most participants meant that the graphical representation was very important for them to catch interest in a game. The overall feeling was that our game's visual presentation was good.

Question 14 could have been split in two parts to give a separate value for first impression, but still, answers indicate that the first impression was good. First impression is an important factor in having the player's curiosity satisfied so that (s)he want to spend more time learning how to play the game in the first place. Impression was that the graphics was well suited for the tasks involved in the game. The game seems to have succeeded in creating a graphical appealing interface, but responses to how fascinated the players were gave slightly lower scores. As the graphical representation is very important, the graphics and animations seen in the game still have room for improvements.

Does the graphical interface contribute to enjoyment? Yes.

9.2.3 GameFlow:Concentration

The GameFlow theory states that a game should require concentration and that the player should be able to concentrate on the game. Questions 17 to 19 are related to this issue.

Participants experienced that tasks they had to concentrate on seemed relevant to finish the game successfully. When asked if the amount of tasks could have been more appropriate, everyone confirmed, except for one participant who disagreed. This could

indicate that more options are needed, like more diversity in hardware and software products.

The overall feeling was that players wanted to concentrate on tasks to solve the objectives. More stimuli (sensory curiosity) and challenging events could improve the overall feeling on concentration.

Does concentration contribute to enjoyment? Yes.

9.2.4 GameFlow:Challenge

Games should be sufficiently challenging and match the player's skill level. Questions 3 to 7 are related to challenge.

In general, participants did not feel that there was too much variation in difficulties. During the analysis it was identified that question three could have been rephrased by excluding "too much" from the statement. This would have made it easier to conclude if the variation was appropriate, too low or too high. Still, results indicate that the players wanted more variation both in the scenarios and during game play.

Another indication that players wanted more variation could be identified in question 5. More variation could be accomplished by designing more scenarios with more objectives and better balanced scenario-settings.

Players agreed that levels were adapted to their prior skills, which enhances the assumption that the game is appropriate for network and network attacks (ref: fantasy).

Some players felt that they mastered the game, at least none disagreed. The feeling of mastering the game could be improved by implementing better feedback mechanisms and progressive levels of difficulties during game play.

Does the challenge contribute to enjoyment? Maybe.

9.2.5 GameFlow:Player Skills

Questions 20 to 22 is related to player skills. GameFlow state that games must support player skill development and mastery. The need to read the user manual should be kept to a minimum.

Feedbacks show that participants needed to read the user manual to understand how to play the game. This could be due to the complexity of the user interface or the player's lack of prior knowledge in this type of game interface. Instructional scenarios could be implemented to make the introduction easier. The game interface could be made even easier to handle by introducing additional helper-functions, wizards and other template mechanisms.

The participants found it interesting to figure out how to play the game, so even if they had to read the user manual, they were motivated to do so. This motivation could be explained by the participant's background and their interest in the game's topic.

The way they felt rewarded by playing successfully seemed unclear. We should put more effort into providing better feedback or other sensory mechanisms to enhance the feeling of reward (or penalty). The score system should be reconsidered using alternative metrics and maybe provide mechanisms so that players get a better sense on how well they are playing during the game instead of presenting a final score when the game has ended.

Does the player skill factor contribute to enjoyment? Maybe.

9.2.6 GameFlow:Control

Control is related to the player's feeling of control over their actions in the game. This is covered by questions 23 to 28.

None disagreed on the statement of mastering the game elements, but responses seemed a bit unclear. The responses were almost identical in question 24, but one participant felt he did not master the graphical interface. To improve the feeling of mastering the game elements and graphical interface, the dialog boxes used to configure these items could be made easier to handle. The tasks of editing network connections, routing tables, installing applications and configuring firewalls should be made simpler to handle. The ability to store firewall filter rules globally should be implemented. An auto-configuration function to ease the task of editing routing tables was implemented, but it could be hard to recognize the correct procedure to benefit from this function. Other approaches, like the use of template-functions for some of these tasks should be considered.

There was a strong feeling that it was not easy to recover from mistakes. This is something that should be solved. The simulation used by the game only works in one time-direction. Storing a number of full states of the complex state model at frequent intervals would be too costly. A way to solve the recovery-problem is to log every action related to the user-interaction. Every logged action, for example adding or connecting a network unit, should have a counter-action, in this case; removing or disconnecting the network unit. By using a log, with a selected number of undo steps, actions could be inverted by removing the last log and executing a counter action. Full state models could be stored by letting the user save his/her game to a file so that (s)he later can restore it by loading the game back from the file.

Players agreed that the actions they performed were meaningful and not boring.

Does the feeling of control contribute to enjoyment? Maybe.

9.2.7 GameFlow:Clear Goals

According to question number one, all players felt that it was easy to understand the game objectives. Goals are presented for each scenario every time the player starts a new game. **Does clear goals contribute to enjoyment? Yes**.

9.2.8 Feedback

Players must receive appropriate feedback at appropriate times (questions 9 to 12).

Some of the participants was negative to the way traffic-sniffing information was presented, and felt that this gave too much feedback. Therefore, players should be able to enable or disable this window at their own discretion.

When asked if feedback helped them achieve their goals, most responses were positive.

Responses were unclear when asked if the feedbacks were surprising. In the game's current state, it is hard to identify how feedbacks could be surprising, but if attack and multi-player options are implemented in the future this could give access to a wide range of surprising events and feedbacks. Another option could be to construct more sophisticated malware tasks.

Most participants felt that feedbacks were constructive. The news-flash function seems to be appropriate in informing the player about important events.

Does feedback contribute to enjoyment? Maybe.

9.2.9 GameFlow:Immersion

Players should experience deep but effortless involvement in the game. Question 29 is related to this issue. Results from this questions does not given any clear indications that players felt any deep involvement. The question could have been a little too vague. More questions could have been added being more direct on the immersion characteristics.

Does the immersion contribute to enjoyment? Maybe.

9.2.10 GameFlow:Competition

Neither a multi-player option nor a global high-score list was implemented, but it was interesting to know, to what extent, a multi-player option could have improved the game's enjoyment level. Everyone meant this would, or absolutely would, have a positive impact on their interest in the game. The current model was designed with a multi-player option in mind. The idea is to let the Internet node manage the WAN objects on a remote game server and let multiple clients connect to this server. Information global to the network security game could be handled by this server. The server would also be responsible for sending packets between networks created by connected players. Additionally, the thread-model could be converted to a linked-list of code instructions, making it possible for players to construct their own set of malwares and exploits used in attacks on other players' networks.

Can competition contribute to enjoyment? Absolutely.

9.2.11 Other results

Results from the open-ended questions indicates that players think the game speed is too high and that events occur too frequently. Also, players feel overwhelmed by the amount of information they receive. Some say that the game speed should be adjustable. This is a good idea and something that should be considered in future versions.

Complaints about the steep learning curve could be solved by introducing scenarios with interactive instructions using animations, voice and/or text windows.

Participants were very positive regarding the graphical interface which they felt resembled "the real thing". Remarks on the positive game properties confirms the game's face and content validity in relation to network and host security.

By showing a prototype of something, this often sparks new ideas in peoples mind. This was also the intention with the open-ended question about ideas to make the game more fun and interesting. A multi-player option is requested by many participants showing that a competitive factor would probably improve the experienced enjoyment quite a lot. Also more sophisticated graphics are requested which resembles the findings in chapter 9.2.2. And not surprisingly, music and sound effects is also something the participants want to see. More product variation is wanted, and another good idea is to have the players construct scenarios and software/hardware products themselves.

9.2.12 Summary

Table 4 represents the results from discussions on enjoyment.

Only two categories, according to GameFlow contributes to the feeling of enjoyment; the ability to concentrate in the game, and the use of game objectives. At the other categories, participants had a more or less neutral attitude. Even though the results could have been better, these are not bad results. Results seems to indicate that players felt some degree of enjoyment, possible due to the use of graphics on which received high

Topic	Fun?
Malone: Fantasy	Yes
Malone: Sensory	Yes
GameFlow: Concentration	Yes
GameFlow: Challenge	Maybe
GameFlow: Player Skills	Maybe
GameFlow: Control	Maybe
GameFlow: Clear Goals	Yes
GameFlow: Feedback	Maybe
GameFlow: Immersion	Maybe

Table 4: Results from the different enjoyment categories

scores. The high score on fantasy could partially explained by the players prior knowledge with computer networks.

The game includes the basic simulation elements; the executive kernel, entities[1] and its relations, distributions, a timer, and the result collector (statistics and triggers). Therefore, it could be classified as a *simulation*. By using entities like virtual users and networked equipment as simulated agents it could also be classified as *agent-based*[32].

Using NarayanaSamy's taxonomy[3], we can say that a virtual environment is present[2]. The user is engaged interactively in the simulation. It provides some entertaining elements, and includes challenges. Game-patterns vary due to the randomness of events occurring in the simulation. The network simulation, upon which the game has been constructed, has no obvious end-state, but goal-oriented scenarios have been introduced to establish end-game objectives. This allows the game to be classified as a *simulation game*.

According to Saunders[5], the game could be classified in the *canned attack/defend scenarios category*.

[1] including agent-entities
[2] the local site network and the Internet

10 Future work

10.1 Determine enjoyment

Due to the low number of participants in this study and because results indicates that the game could be fun to play, a new study should be conducted. It is recommended that the game is enhanced with new challenges and and sensory items like better graphics and sounds before such a study takes place.

10.2 Additional Features

Additional malware tasks should be constructed to take full advantage of the attack simulation and add new challanges to the game. New network equipment should be added to the model, like NAT routers, stateful firewalls, VPN concentrators, modems, VLANs, wireless units, etcetera. This would give the player more options to design a secure network. A new study could be conducted to map these nodes into the existing model and by doing so, increasing the model's diversity.

10.3 Validity as a Learning Tool

The game should be validated as a learning tool. A set of documents to teach different aspects of network security should be constructed and game scenarios related to this learning material should be developed. A randomized pretest-post-test control group design, with one group using the learning material only, another group using the game only, and a third group using both instruments, could be conducted to see if the game has any positive impact on the learning process.

10.4 Multiplayer gaming

It should be investigated if the model can be split into a client-server model. An idea is to move the Internet-node to a separate server and have multiple clients connected to this server. As seen in this study, a multi-player version could increase the interest in the game.

10.5 Dynamic Construction

The process-thread mechanism could be reorganized so that threads are executed in a linked list of code instructions. This way, players should be able to construct their own algoritms and malwares and use their algoritms to attack remote sites played by other clients connected to the game server. Some work is needed to figure out how to accomplish this. A multi-player model could also increase the random behavior of networks as real networks are involved in the transmission of simulated packets.

10.6 Fine tuning

In the current version, the game model depends heavily on the construction and destruction of .NET objects. These are CPU expensive operations. One issue is the use of the .NET garbage collector. In simulated networks with a lot of traffic, a large amount

of packets are created and destroyed. A change in the model could rely more on global memory structures and arrays using memory pointers.

10.7 Better Graphics

The graphic handling is separated from the simulation engine. A new graphic handler based on OpenGL or DirectX could be created to improve performance and animations. As seen in this study, the use of graphics to enhance the fantasy.

11 Conclusion

Having received feedbacks from participants with prior skills in network administration and network security and some with many years experience, results from the experiment shows that the game should be valid for network security gaming according to the fantasy measurements and open-ended feedbacks related to game properties.

This study concludes that it is possible to build a network and host security game, with the same complexity as seen in this study, in five months on the assumption that the developer have prior skills in network and host security, multi-threading programming, object oriented analytical skills, and know-how in building advanced graphical editors. The developer must also expect to use more than 1200 man-hours to complete the game within this time period. A copy of the final prototype can be downloaded from `http://www.stud.hig.no/(tilde)040673`. This concludes the first research question.

The degree of enjoyment, does not seem to be high enough for people to keep playing the game for a longer period of time. Further studies must be conducted with more participants to tell with enough degree of confidence that a network security game is fun to play. But first, the game should be extended with more challenges and fantasy-enhancing features like better graphics, music and sound effects. In addition, further studies should focus on competitive elements, like multi-player and real-time attack/defend scenarios, to make it even more enjoyable.

There are indications that people with prior skills finds the game fun to play, to some extent. But further studies must be conducted with enough participants to confirm this by statistical means. This concludes the second research question.

Bibliography

[1] C.E.Irvine & M.Thompson. 2003. Teaching objectives of a simulation game for computer security. http://cisr.nps.navy.mil/downloads/03paper_cciege.pdf.

[2] C.E.Irvine, M. & K.Allen. 2005. Cyberciege: An information assurance teaching tool for training and awareness. http://cisr.nps.navy.mil/cyberciege/downloads/FISSEA_CyberCIEGE_PreConf.pdf.

[3] V. Narayanasamy, K. W. Wong, C. C. F. & Rai, S. 2006. Distinguishing games and simulation games from simulators. *ACM Computers in Entertainment*, 4(2).

[4] M.Prensky. 2002. Why not simulation. http://www.marcprensky.com/writing/Prensky%20-%20Why%20NOT%20Simulation.pdf.

[5] Saunders, J. H. 2002. Simulation approaches in information security education. http://cisse.info/CISSE%20J/2002/saun.pdf.

[6] J.Saltzer & M.Schroeder. 1975. The protection of information in computer systems. *Proceedings of the IEEE*, 4(2), 1278–1308.

[7] M.Bishop. 2002. *Computer Security: Art and Science chap 13*. Addison-Wesley.

[8] G.Stoneburner, C. & A.Feringa. 2001. Engineering principles for information technology security (a baseline for achieving security. *National Institue of Standard and Technology Special Publication 800-27*.

[9] International Standard. *ISO/IEC 17799 - Information Technology - Code of practice for information security management*, 1 edition, 12 2000.

[10] 2005. Iso 20000.

[11] Landwehr, Bull, M. & Choi. 1994. A taxonomy of computer program security flaws. *ACM Computing Surveys*, vol 26, Issue 3 (september 1994)(0360-0300), 211 – 254.

[12] Criteria, C. Common criteria for information technology security evaluation. http://www.commoncriteriaportal.org/public/consumer/index.php?menu=2.

[13] N.Weaver, V.Paxon, S. & R.Cunningham. 2003. A taxonomy of computer worms. http://www.cs.berkeley.edu/ñweaver/papers/taxonomy.pdf.

[14] T.W.Malone. 1980. What makes things fun to learn? heuristics for designing instructional computer games. *Proceedings of the 3rd ACM SIGSMALL Symposium and the First SIGPC Symposium on Small Systems, Palo Atlo, California, USA, ACM Press*, 162–169.

[15] Sweetser, P. & Wyeth, P. 2005. Gameflow: A model for evaluating player enjoyment in games. *ACM Computers in Entertainment*, 3(3).

[16] Csikszentmihalyi, M. 1990. *Flow: The Psychology of optimal experience*. Harper Perennial; Rep edition (March 13, 1991).

[17] K.Näckros. 2002. Empowering users to become effective information security and privacy managers in the digital world through computer games. http://smg.media.mit.edu/cscw2002-privacy/submissions/kjell.pdf.

[18] M.Prensky. 2004. *Digital Game-Based Learning*. Paragon House, 1 edition.

[19] B.Schneier. 2000. *Secrets and Lies*. Wiley.

[20] Group, O. M. 2003. Unified modeling language specification. http://www.omg.org/cgi-bin/doc?formal/03-03-01.

[21] E.Gamma, R.Helm, R. & H.Vlissides. 2000. *Design Patterns - Elements of Reusable Object-Oriented Software*. Addison-Wesley.

[22] C.L.Owen. 1998. Design research - building the knowledge base. *Design Studies*, No.1 Jan. 1998, 9–20.

[23] D.Turner. 2005. Symantec internet security threat report. trends for january 05-june 05.

[24] B.A.Forouzan. 2004. *Data Communications and Networking*. McGraw Hill, 3 edition.

[25] W.R.Cheswick & S.M.Bellovin. 1994. *Firewalls and Internet Security*. Addison-Wesley, 3 edition.

[26] K.Kent, S. L. S. & R.W.Ritchey. 2005. *Inside Network Perimeter Security*. Sans Institute, 2 edition.

[27] J.Blow. 2004. Game development. harder than you think.

[28] C.Petzold. 2002. *Programming Microsoft Windows with C#*. Microsoft Press.

[29] E.White, C. & S.Robinson. 2002. *GDI+ Programming: Creating Custom Controls using C#*. Wrox.

[30] P.Drayton, B. & T.Neward. 2003. *C# In a Nutshell. A Desktop Quick Reference*. O'Reilly.

[31] A.L.Dennis. 2003. *.NET Multithreading*. Manning Publications, 1 edition.

[32] S.M.Sanchez & T.W.Lucas. 2002. Exploring the world of agent-based simulations: Simple models, complex analysis. *Proceedings of the 2002 Winter Simulation Conference*, 116–126.

[33] Williams, R. 2001. *Computer System Architecture. A Networking Approach*. Addson-Wesley.

[34] W.R.Stevens. 1994. *TCP/IP Illustrated Volume 1 - The Protocols*. Addison-Wesley.

[35] J.F.Kurose & K.W.Ross. 2003. *Computer Networking. A Top Down Approach Featuring the Internet*. Addison Wesley).

[36] J.Postel. 1981. Transmission control protocol - darpa internet program protocol specification std7 rfc793.

[37] J.Viega & G.McGraw. 2001. *Building Secure Software*. Addison-Wesley.

[38] C.E.Landwehr & D.Goldschlag. 1997. Security issues in networks with internet access. *Proceedings of the IEEE*, Volume 85, Issue 12, Dec. 1997, 2034–2051.

[39] J.Postel. 1980. User datagram protocol std6 rfc768.

A Feedback questionnaire

Age: years
Sex: Female or Male
Education: select one of the following;

- Primary School - Elementary
- Middle School - Junior High
- Secondary School - High School/Gymnasium
- Post-secondary - University/College
- Post-graduate - Masters degree
- PhD or higher

Experience with network administration (years):
Experience with network security (years):

A.1 Statements and responses

The following statements must be answered with one of the following responses;

- 1 = Not at all
- 2 = No
- 3 = Maybe (neutral)
- 4 = Yes
- 5 = Absolutely
- (blank) = Don't know

1. **It was easy to understand the game objectives**:

---

2. **The outcome was sure when I played the game**:

---

3. **There was too much variation in levels of difficulty**:

---

4. **Levels of difficulties was adapted to my prior skills**:

---

5. **The game soon became boring due to lack of levels of difficulty**:

---

6. **There was too much randomness in the game**:

---

7. **I got I feeling that I mastered the game**:

---

8. **I managed to identify in the role as a network administrator**:

---

9. **The amount of feedback received fitted the purpose of the game**:

10.**Feedbacks helped me find solutions to succeed in reaching the goal successfully**:

11.**Feedbacks was surprising**:

12.**Feedbacks was constructive**:

13.**The graphical representation means a lot to me to catch my interest in games**:

14.**The game's visual representation was good. It got me interested at first-sight**:

15.**The visual representation was nicely adapted to the tasks I had to conduct**:

16.**I was fascinated by the graphical representation**:

17.**The tasks I had to do seemed to be relevant to finish the game successfully**:

18.**I got a feeling that I wanted to concentrate on the tasks to solve the objectives**:

19.**The amount of tasks I had to do was appropriate**:

20.**I did not have to read the user manual to understand how to play the game**:

_____

21.**It was interesting to find out how to play the game**:

_____

22.**I was well rewarded by playing successfully**:

_____

23.**I got a sense that I mastered the elements I had to work with**:

_____

24.**I got a sense that I mastered the graphical game interface**:

_____

25.**I was easy to recover from errors made during the game**:

_____

26.**I got a sense that the tasks I conducted was meaningful**:

_____

27.**Tasks were boring**:

_____

28.**I got a sense that the tasks I had to do were predetermined. I had few alternatives to solve the objectives**:

_____

29.**I felt a deep involvement in the game**:

_____

30.**I think the concept of the game is good and well suited for network security**:

_____

31.**I think the game can motivate me to learn more about network security on my own**:

_____

32.**If this game was used in a learning environment, together with literature, it would have a very positive impact on my motivation to learn more about network security**:

_____

33.**It is important for me to share my game experience, what I construct, and my results with a game community**:

_____

The next questions relates to multi-player gaming. There is a theory that this game could be constructed as a multi-player game with multiple players having their networks connected via a central game-server acting as the Internet node.

34. **This would have a positive impact on my interest for the game**:

_____

35. **This would have a positive impact on my motivation to learn more about network security on my own**:

_____

A.2 Open ended questions

What do you think was the negative properties of the game?

_____

What was the most negative property of the game?

_____

What do you think was the positive properties of the game?

_____

What was the most positive property of the game?

_____

Do you have some ideas on how to make the game more fun and interesting to play?

_____

How could the game make you more motivated to learn more about network security?

_____

Other comments:

_____

B Simposter User manual

B.1 About the game

Simposter is a game based on a simplified network simulator. Data packets are sent back and forth between applications on different hosts connected together via the local- or Internet- network. Like in real life, packets need to be routed to get from one host to another (and back again) and firewalls need to be configured to stop traffic from reaching protected services.

Traffic is generated by virtual users using applications installed on their client computers. These users have different characteristics and moods that influence the type of applications they prefer to use and the activities they conduct.

B.2 Game Objectives

You are the administrator of a local network. The objectives of this game is to create an effective, functional and secure network. Documents (assets) created or downloaded by network users must be protected from confidentiality and integrity breaches. You have different tools at your disposal, like different types of network components (switches, routers, firewalls) and software tools (backup, anti-virus, spam-filter, office-tools and various server applications). How you design your network, what software you choose to install and how hardware and software are configured will have an impact on your final score. Everything comes at a cost. Your resource management skills are important to achieve high scores.

B.3 Network design

The network map

Network design is an important game element in Simposter. The main screen displays a two-dimensional map with cells representing the components of your local network. To keep you oriented, a vertical and horisontal coordinate system is displayed on the left side and at the top of the map view. To scroll, use the arrow keys on your keyboard, or left-click on a free cell and move your mouse in any direction.

B.4 Mouse operations

Right-click on any component to display the short-cut menu. Short-cut menus makes it easier to access the functionality and properties of network components.

Simposter has five different mouse operation modes - select, move, scroll, connect and delete. The select-mode is the base mode of all operations. To cancel any operation other than select, press the CTRL key on your keyboard. If you are in select-mode, pressing (and holding) the CTRL key will enter the connect-mode. While in connect-mode, each component shows a variable number of dots. These are the interfaces (network card/-port) for each component. Dots with a light green color is already connected to another component. If all dots show a light-green color, there are no free interfaces left on this component. To connect, select the node you want to connect from (typically a switch).

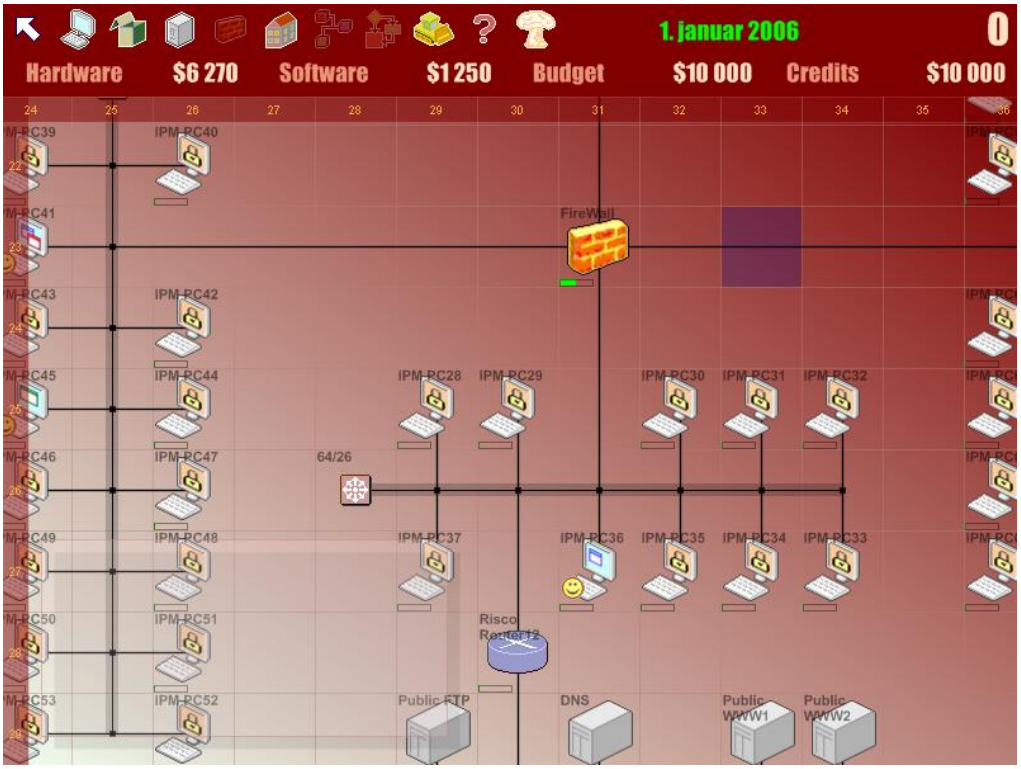

Figure 35: Simposter Game Screen

Press, and hold, the CTRL key and click on each component you want to connect.

Tip: If you have created a new switch (sub-net) and numerous clients, select the switch first, then press CTRL and click on each client in sequence. Clicking a new node while in connect-mode will not select the new node.

B.5 Key Controls

B - Buy new hardware item

DEL - Sell selected component

CTRL (tap once) - Enter the select-mode (mouse)

CTRL (hold) - Enter the connect-mode (mouse)

ESC - Close dialog box / abort game

B.6 The tool bar

This is a description of the tool-bar buttons (from left to right)

Button Key Description Select Tool CTRL Select this button (or press CTRL) to enter select-mode. In select mode, existing components can be selected and moved. To move an item, left click to select an item and drag it to the cell where you want to have it. Buy Hardware B Select this button (or press B) to buy new hardware components (clients, servers, firewall, etc) Software Management Select this button to buy or un-install software on multiple clients/servers simultaneously. Server Management Select this button to manage your servers and set public/private services. FireWall Configuration If you have selected a firewall component this button will be enabled and you will be able to

change the firewall's filter rules. Scenario Opens a dialog box that shows you the game level objectives and game configurations Connect Clicking this button will enable the special connect mouse-mode where you will not need to press and hold the CTRL key to connect. Cancel this mode by pressing the select-tool or press CTRL. Show Component Press this button to display the property dialog for the selected component. The property dialog can be used to configure items, route tables, applications (client/server components) and show process, connection and program/asset status. You can also double-click on any node to display this dialog. Delete Press this button to enter the delete/sell mouse-mode. Click on any component to delete/sell it. Cancel this operation by pressing the select-tool or press the CTRL key. Help Click this button to show this user manual Quit Press this button to abort the game.

B.7 Route table/Sub-net configuration

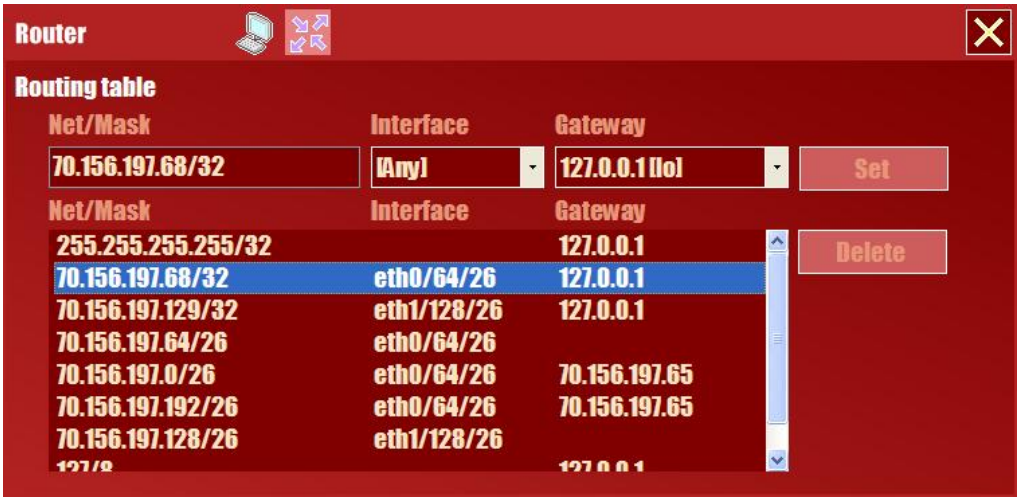

Figure 36: Simposter Routing table Screen

A routing table can be configured for routers, firewalls, clients and servers. Use this table to register local IP addresses, interfaces, sub-nets and default gateways (0/0). The network simulation uses this information to route packets between two hosts.

B.8 Sub-net configuration

The process of creating routing tables are semi-automated in Simposter. Switches (net-segments) have an alternative registration form where sub-net ranges can be registered, so that subsequent client connections will receive available IP-addresses from these ranges.

A range can be changed at any time. After changing the sub-net address range, right-click on the net-segment and select to auto-configure client connections. This will update the IP-addresses of connected components (this will change the routing table configuration for connected components). The local sub-net address range(s) is/(are) the one with no registered gateway. The other registrations are used by connected routers (routing information between local sub-nets).

Figure 37: Simposter Sub net dialog box

B.9 Setting up firewall rules

Use the firewall to create a chain for rule filters to allow or deny inbound and/or outbound packets on any interface.

B.10 Installing software

Software can be installed in two ways; single host or batch-install on multiple hosts. Single host installation can be achieved by right clicking the host component and select Install -> Software Category -> Product from the pop-up menu. Batch-install can be done through the Software Managment dialog box available from the game toolbar.

B.11 Creating public services

By default, when a server application is installed, only local network users are aware of this service - only local users will try to connect to these services (be aware that remote attackers might also attempt to connect to these services!!!).

To make a private service public (like public web service, FTP service, etc), enter the server managment dialogbox from the toolbar.

On the left site, only internal (private) services are shown as service and machine name. Private/public services will not be influenced by changes in host IP-adresses (that's why the name, and not the IP address is shown in this dialog).

At lower edge of the dialogbox, the default protocol server of selected protocols can be changed. This will change the configuration on already installed client applications and it will be the default value for new installations.

By default, the default DNS, POP3 and SMTP server will set to the ISP's (WAN components') IP address.

DNS, POP3 and SMTP services can also be installed in the local network. Then, change the default server to this/these hosts.

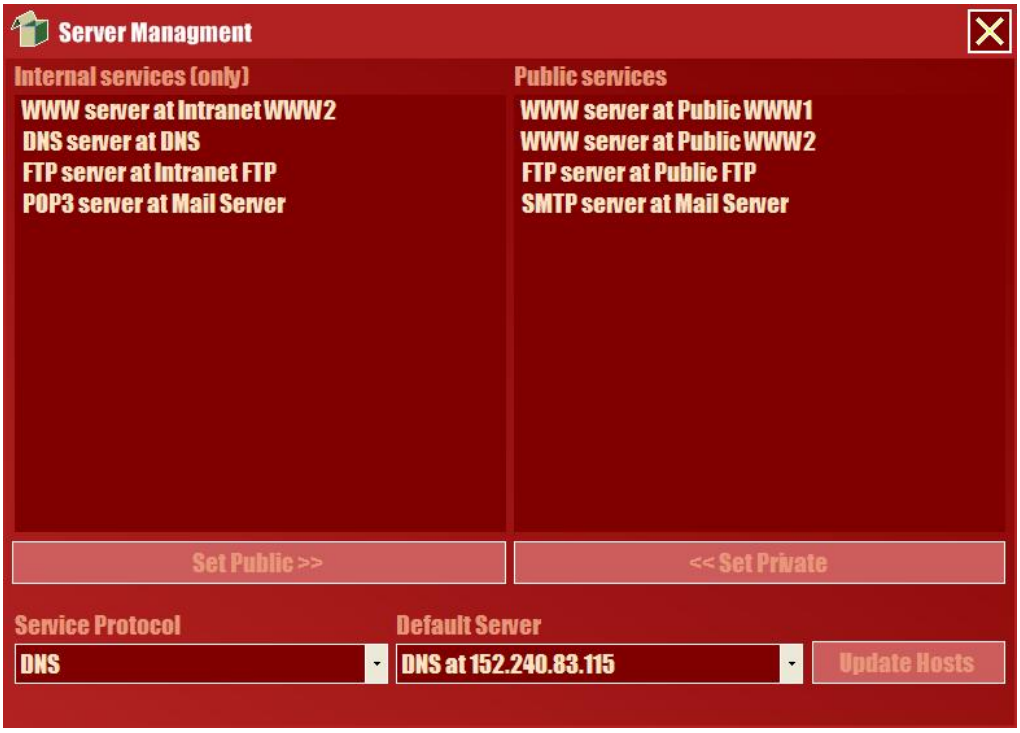

Figure 38: Simposter Server Management dialog box

B.12 Hints and tips

Ensure that all public (and private) services has enough capacity to handle requests. Select a product and read the description to understand its capabilities.

To receive mails locally, POP3 and SMTP must be installed at the same host and the SMTP server must be set public. Having a local mail server makes it possible to install spam-filters thus increasing user productivity.

Add a SQL server service and configure Web (HTTP) servers to use this SQL server. This will improve the network functionality and you will get higher scores.

Listen to user requests.

Install anti-virus software to reduce the damage caused by malware.

Remember to patch software products to reduce vulnerability and malware infections.

Use a remote file server. Install a backup tool on this server. You will get more points by using a centralized backup facility instead of backing up data locally (less secure).

Keep you network secure and functional the best you can - Good luck!

C Statistical enumerators

```
public enum StatTypes : int {
        GAME_Failures,
        GAME_Successes,
        GAME_EndFailure,
        GAME_EndSuccess,

        USER_PowerUps,
        USER_PowerDowns,
        USER_NoPowerDowns,
        USER_Logins,
        USER_Logouts,
        USER_ForgotLogout,
        USER_EnterComputerOwn,
        USER_EnterComputerOther,
        USER_UsesWrongLogin,
        USER_LogoutOther,
        USER_StartsWorking,
        USER_LeavingComputer,
        USER_WasLoggedInForgotLogout,
        USER_TerminalOccupied,
        USER_ClaimsNoComputer,
        USER_Deleted,
        USER_Created,
        USER_IntranetNotFoundComplaints,

        NODE_ClientsAdded,
        NODE_ServersAdded,
        NODE_HubsAdded,
        NODE_SwitchesAdded,
        NODE_RoutersAdded,
        NODE_FirewallsAdded,
        NODE_WANsAdded,
        NODE_ClientsRemoved,
        NODE_ServersRemoved,
        NODE_HubsRemoved,
        NODE_SwitchesRemoved,
        NODE_RoutersRemoved,
        NODE_FirewallsRemoved,
        NODE_WANsRemoved,

        HOST_DataFilesDownloaded,
        HOST_ProgramFilesDownloaded,
        HOST_ProgramsInstalled,
        HOST_ProgramsInstalledMalware,
        HOST_ProgramsUninstalled,
        HOST_DataFilesCreated,
        HOST_ForgottenFormatsBeforeSale,
```

```
LINK_Connected ,
LINK_Disconnected ,

FILE_DataFilesSaved ,
FILE_DataFilesExploitSaved ,
FILE_DataFilesDeleted ,
FILE_DataFilesShadowed ,
FILE_DataFilesShadowDeleted ,
FILE_ProgramFilesSaved ,
FILE_ProgramFilesMalwareSaved ,
FILE_ProgramFilesDeleted ,
FILE_ProgramFilesShadowed ,
FILE_ProgramFilesShadowDeleted ,
FILE_ProgramFilesOpened ,
FILE_DataFilesOpened ,

SAMBA_ClientPutFileRequests ,
SAMBA_ClientGetFileRequests ,
SAMBA_ClientFilesReceived ,
SAMBA_ClientPutFileSucceeded ,
SAMBA_ClientGetFileSucceeded ,
SAMBA_ClientGetFileFailed ,
SAMBA_ClientPutFileFailed ,
SAMBA_ClientMissingRemoteAddress ,
SAMBA_ServerNoFilesAvailable ,
SAMBA_ServerSaveFileFailed ,

REMOTEBACKUP_Files ,
REMOTEBACKUP_Count ,
REMOTEBACKUP_Failed ,
LOCALBACKUP_Files ,
LOCALBACKUP_Count ,

BACKUPTOOL_Backups ,
BACKUPTOOL_DocumentsStored ,
BACKUPTOOL_ProgramsStored ,

ANTIVIRUS_ProgramFilesScanned ,
ANTIVIRUS_DataFilesScanned ,
ANTIVIRUS_DataFilesMatch ,
ANTIVIRUS_DataFilesFalseMatch ,
ANTIVIRUS_ProgramFilesMatch ,
ANTIVIRUS_ProgramFilesFalseMatch ,
ANTIVIRUS_DownloadedProgramsDeleted ,
ANTIVIRUS_VirusDeletedAndFileOpened ,
ANTIVIRUS_ProcessesInfected ,
ANTIVIRUS_ProcessesInfectedMissingSignature ,
ANTIVIRUS_ProgramFilesMissingSignature ,
ANTIVIRUS_DataFilesMissingSignature ,

EXPLOIT_ExploitsDuringOpen ,
EXPLOIT_FileExploitsActivated ,
```

EXPLOIT_PacketExploitsActivated ,
EXPLOIT_NotOpenedProgramPatched ,

MALWARE_MailsSent ,

SCRIPTKID_Attacks ,
SCRIPTKID_DNSLookups ,
SCRIPTKID_ConnectedToFTP ,
SCRIPTKID_ConnectedToPOP3 ,
SCRIPTKID_ConnectedToSAMBA ,
SCRIPTKID_ConnectedToSMTP ,
SCRIPTKID_ConnectedToSQL ,
SCRIPTKID_ConnectedToHTTP ,
SCRIPTKID_SuccessfulAttacks ,

FTP_ProgramFilePutRequests ,
FTP_DataFilePutRequests ,
FTP_FileGetRequests ,
FTP_RemoteSiteSuccessfullyDownloadedFile ,
FTP_ProgramFilePutSucceeded ,
FTP_DataFilePutSucceeded ,
FTP_ProgramFileDownloadedMalware ,
FTP_ProgramFileDownloaded ,
FTP_DataFileDownloadedExploit ,
FTP_DataFileDownloaded ,

MAIL_GetMailRequests ,
MAIL_GetMailRequestsSucceeded ,
MAIL_MailsReceived ,
MAIL_MailsWithAttachments ,
MAIL_AttachmentsReceived ,
MAIL_ProgramFilesReceivedMalware ,
MAIL_ProgramFilesReceived ,
MAIL_DataFilesReceivedExploit ,
MAIL_DataFilesReceived ,
MAIL_GetMailRequestsReturnedNoMails ,
MAIL_SpamMails ,

SPAM_AcceptedFalsely ,
SPAM_Rejected ,
SPAM_RejectedFalsely ,

MAIL_SendMailRequests ,
MAIL_MailsSent ,
MAIL_SendMailAttachments ,

POP3_MailRequests ,
POP3_MailsReturnedToUser ,
POP3_NoMailsReturnedToUser ,

SQLCLIENT_Requests ,
SQLCLIENT_RequestsSucceeded ,
SQLSERVER_RequestsHandled ,

HTTP_UrlRequests ,
HTTP_ProgramFilesDownloadedMalware ,
HTTP_ProgramFilesDownloaded ,
HTTP_DataFilesDownloadedExploit ,
HTTP_DataFilesDownloaded ,

HTTPSERVER_RequestsHandled ,
HTTPSERVER_SqlRequests ,
HTTPSERVER_SqlRequestsFailed ,
HTTPSERVER_FilesSent ,

HOST_FormatsInitiated ,
HOST_FormatsSucceeded ,

PROCESSES_Created ,
PROCESSES_Terminated ,
PROCESSES_TerminatedCrash ,
PROCESSES_TerminatedSystemShutdown ,
PROCESSES_TerminatedUninstall ,
PROCESSES_TerminatedUserShutdown ,
PROCESSES_CreatedWithMalware ,

THREADS_Created ,
THREADS_Exceptions ,
THREADS_Terminated ,

ROUTING_RoutesAddedStatic ,
ROUTING_RoutesAddedDynamic ,
ROUTING_RoutesDeletedStatic ,
ROUTING_RoutesDeletedDynamic ,
ROUTING_PacketsForwarded ,
ROUTING_DropProcessedBySystem ,
ROUTING_DropSocketBufferFull ,
ROUTING_DropSwitchCantRoute ,
ROUTING_DropHostCantRoute ,
ROUTING_DropAddressNotFound ,
ROUTING_DropTimedOutPending ,
ROUTING_DropRouteNotAvailable ,
ROUTING_DropTooManyHops ,
ROUTING_DropNotExpectedInCurrentState ,
ROUTING_DropSocketTerminated ,
ROUTING_DropMaxConnectionRequestsReached ,
ROUTING_DropOther ,
ROUTING_DropDeniedByFirewall ,
ROUTING_DropTotal ,

FIREWALL_FiltersAdded ,
FIREWALL_FiltersUpdated ,
FIREWALL_FiltersRemoved ,
FIREWALL_PacketExploitsStoppedOutbound ,
FIREWALL_PacketExploitsStoppedInbound ,
FIREWALL_PacketsCheckedInbound ,

FIREWALL_PacketsCheckedOutbound ,
FIREWALL_PacketsOutboundDenied ,
FIREWALL_PacketsInboundDenied ,

DNSCLIENT_Requests ,
DNSCLIENT_AddressFound ,
DNSCLIENT_AddressNotFound ,

DNSSERVER_AddressesFound ,
DNSSERVER_AddressesNotFound ,

SOCKETS_UDP_Created ,
SOCKETS_UDP_Registered ,
SOCKETS_UDP_Unregistered ,
SOCKETS_UDP_PacketDataSent ,
SOCKETS_UDP_PacketDataReceived ,

SOCKETS_BoundToLocalPort ,
SOCKETS_BindFailed ,
SOCKETS_PacketsReadByProcess ,

SOCKETS_TCP_Created ,
SOCKETS_TCP_Registered ,
SOCKETS_TCP_Unregistered ,
SOCKETS_TCP_ConnectRequests ,
SOCKETS_TCP_ConnectionRequestAttempts ,
SOCKETS_TCP_ConnectionsRequested ,
SOCKETS_TCP_ConnectionResets ,
SOCKETS_TCP_ConnectionResetsByRemote ,
SOCKETS_TCP_ConnectionsAccepted ,
SOCKETS_TCP_ConnectionRequestsLimitReached ,
SOCKETS_TCP_Listen ,
SOCKETS_TCP_ListenSucceeded ,
SOCKETS_TCP_PacketsSent ,
SOCKETS_TCP_PacketsReceived ,
SOCKETS_TCP_PacketDataSent ,
SOCKETS_TCP_PacketDataReceived ,
SOCKETS_TCP_HandshakesCompleted ,

MONEY_Amount ,
MONEY_MonthlyBudget ,
MONEY_MoneyUsed ,
MONEY_MoneyUsedPenalty ,
MONEY_SpentHardwareInstall ,
MONEY_SpentSoftwareInstall ,
MONEY_SpentDailyHardware ,
MONEY_SpentDailySoftware ,
MONEY_PatchCost ,

SCORE_Points ,
SCORE_FinalScore ,
SCORE_Confidentiality ,
SCORE_Integrity ,

```
        SCORE_Availability,
        SCORE_Penality,
        SCORE_Bonus,

        SCENARIO_ConfidentialityImportance,
        SCENARIO_IntegrityImportance,
        SCENARIO_AvailabilityImportance,
        SCENARIO_InitialMoney,
        SCENARIO_MonthlyBudget,
        SCENARIO_SitePopularity,
        SCENARIO_MalwareFrequency,
        SCENARIO_LocalUserSpeed,
        SCENARIO_RemoteUserSpeed,
        SCENARIO_SpamFrequency,
        SCENARIO_RecruitmentFrequency,
        SCENARIO_RetirementFrequency,
        SCENARIO_RemoteMailFrequency,
        SCENARIO_IntranetRequests,

        ENGINE_Time,
        ENGINE_NodesExecuted,
        ENGINE_ThreadRoundtrips,
        ENGINE_UserOperationsExecuted,
        StatTypesLast
};
```

D The simulation Loop

```
/// <summary>
/// Main loop for each thread
/// </summary>
void __SimulationLoop( object threadInfo ) {

  ThreadInfo info = (ThreadInfo)threadInfo;
  List<Executable> threadNodes = info.Executables;
  int threadIndex = info.ThreadIndex;

  Executable[] simnodes = threadNodes.ToArray();

  // Store initial number of executables
  int count = simnodes.Length;

  // Enter loop. Execute loop while simulation is running
  while ( _running ) {
    // Compare number of old and current items.
    // Update thread executable collection
    // if size has changed (a node has been
    // added/removed externally)
    lock ( threadNodes ) {
      if ( count != threadNodes.Count ) {
        simnodes = threadNodes.ToArray();
        count = simnodes.Length;
      }
    }

    // Execute each node in the collection
    for ( int i = 0; i < count; i++ ) {
      simnodes[i].Execute();
      System.Threading.Thread.Sleep( 0 );
    }

    // Increase time and sleep for a few milliseconds
    if ( threadIndex == 0 ) {

      DateTime oldDate = CurrentDate;

      _time++;
      _dayChanged = ( _time % TICKS_PR_DAY ) == 0;

      if ( _dayChanged ) {
        DateTime newDate = CurrentDate;
        _stat.Accumulate(
          StatTypes.MONEY_SpentDailyHardware,
          (int)( _internet.LocalSite.CurrentHardwareCost / 30 ) );
```

```
        _stat.Accumulate(
          StatTypes.MONEY_SpentDailySoftware,
          (int)( _internet.LocalSite.CurrentSoftwareCost / 30 ) );

        _stat.Accumulate( StatTypes.MONEY_Amount,
          -(int)( ( _internet.LocalSite.CurrentHardwareCost +
          _internet.LocalSite.CurrentSoftwareCost ) / 30 ) );

        __CheckTriggers( oldDate, newDate );

      }

      if ( _time % 20 == 0 ) {

        _stat.SetValue(
          StatTypes.SCORE_Points,
          _internet.LocalSite.Score );

        _stat.SetValue(
          StatTypes.ENGINE_Time,
          (int)_time );

        lock ( _triggers ) {
          for ( int i = 0; i < _triggers.Count; i++ ) {
            if ( !_triggers[i].HasBeenTriggered ) {
              int statVal = _stat.GetValue( _triggers[i].TriggerType );
              if ( _triggers[i].CompareType == CompareTypes.LargerThan ) {
                if ( statVal > _triggers[i].TriggerValue ) {
                  _stat.Accumulate(
                    _triggers[i].StatisticsToAccumulate,
                    _triggers[i].StatisticsValue );

                  _triggers[i].HasBeenTriggered = true;
                }
              } else {
                if ( statVal < _triggers[i].TriggerValue ) {
                  _stat.Accumulate(
                    _triggers[i].StatisticsToAccumulate,
                    _triggers[i].StatisticsValue );

                  _triggers[i].HasBeenTriggered = true;
                }
              }
            }
          }
        }
        if ( _stat.GetValue( StatTypes.GAME_EndFailure ) > 0 ||
          _stat.GetValue( StatTypes.GAME_EndSuccess ) > 0 ) {
            _endGame = true;
            _running = false;
        }
      }
    }
```

```
    System.Threading.Thread.Sleep( 20 );

    // Add thread round-trip statiscs
    Accumulate( StatTypes.ENGINE_ThreadRoundtrips );
  }
}
```

E Code sample - Web browser client

```
/// <summary>
/// The Web Client task procedure found in the WebClient application class
/// </summary>
/// <param name="thread">The Task thread</param>
/// <returns>True if code was executed, false if out-of-sequence</returns>
public override bool TaskProcedure( SimThread thread ) {

  // Uses the thread-state variable RunPointer to determine which
  // execution sequence to run in this iteration.
  // Threads are executed in a round-Robin fashion. For each iteration
  // a small portion of a procedure will be executed. This function
  // defines this chunks using a switch-instruction and the
  // current RunPointer.

  switch ( thread.RunPointer ) {

    case 1:  // This (1) is the default entry point for all new threads

      // Create session socket (TCP). Execute line 10 on next execution

      thread.SocketCreate( Protocols.TCP, 10 );
      return true;

    case 10:

      // Get remote web address from thread task
      // and connect to that address. Move run-pointer to line 20
      // when the connection has been established. SocketConnect
      // will block this thread until connection has finished

      object address = thread.Task.Parameters[0];
      ushort remotePort = (ushort)thread.Task.Parameters[1];
      thread.SocketConnect( address, remotePort, Scenario.DEFAULT_TIMEOUT, 20

      return true;

    case 20:

      // Send page request and move run-pointer to line 30

      thread.Host.Accumulate( StatTypes.HTTP_UrlRequests );
      thread.SocketSendPacket( "GET /SOMEDATA", null, 30 );

      return true;

    case 30:
```

```
    // Read reply and add result to the task

    PacketData response =
       (PacketData)thread.SocketRead( Scenario.DEFAULT_TIMEOUT, true );
    if ( response != null ) {
      if ( response.Content != null ) {
        SimFile file = (SimFile)response.Content;
        if ( file is SimFileProgram ) {
          if ( ( (SimFileProgram)file ).Malware != null )
            thread.Host.Accumulate(
              StatTypes.HTTP_ProgramFilesDownloadedMalware );
          else
            thread.Host.Accumulate(
              StatTypes.HTTP_ProgramFilesDownloaded );
        } else if ( file is SimFileData ) {
          if ( ( (SimFileData)file ).Exploit != null )
            thread.Host.Accumulate(
              StatTypes.HTTP_DataFilesDownloadedExploit );
          else
            thread.Host.Accumulate(
              StatTypes.HTTP_DataFilesDownloaded );
        }
      }

      // Move run-pointer to line 31 if web client is running
      // on a host at the local site, otherwise move to line 40

      if ( thread.Host.Site.IsLocal ) {
        // Store any reply content (html-file) in memory
        thread.SetVariable( "result", response.Content );
        thread.Task.Result = response.Content;
        thread.Goto( 31 );
      } else {
        // Remote Site, succeeded in download web page
        thread.Goto( 40 );
      }
    }

    return true;

  case 31:

    // Read the reponse-content from memory and store in
    // ``documents''.

    SimTask task = thread.Save( thread.GetVariable( "result" ));
    if ( task != null ) {
      // The save task finished. Continue to line 40
      thread.Task.Result = task.Result;
      thread.Goto( 40 );
    }

    return true;
```

```
    case 40:

      // Close the socket session. Move to line 50 when the
      // session has been closed. SocketClose will block this thread
      // until the socket has been closed or failed due to a timeout.

      thread.SocketClose( 50 );

      return true;

    case 50:

      // Exit and terminate the thread

      thread.Return();

      return true;
  }

  return false;

}
```