A METHODOLOGY FOR CYBERCRAFT REQUIREMENT DEFINITION AND INITIAL SYSTEM DESIGN

GRADUATE RESEARCH PAPER

Michael G. Hunsberger, Major, USAF

AFIT/ICW/ENG/08-03

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

The views expressed in this graduate research project are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the U.S. Government.

AFIT/ICW/ENG/08-03

A METHODOLOGY FOR CYBERCRAFT REQUIREMENT DEFINITION AND INITIAL SYSTEM DESIGN

GRADUATE RESEARCH PAPER

Presented to the Faculty

Department of Electrical & Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Cyber Warfare

Michael G. Hunsberger, BS, MS

Major, USAF

June 2008

AFIT/ICW/ENG/08-03

A METHODOLOGY FOR CYBERCRAFT REQUIREMENT DEFINITION AND INITIAL SYSTEM DESIGN

Michael G. Hunsberger, BS, MS
Major, USAF

Approved:

/signed/	10 June 2004
Lt Col J. Todd McDonald, USAF, Ph.D. (Chairman)	Date
/signed/	10 June 2004
Dr. Robert F. Mills (Member)	Date

AFIT/ICW/ENG/08-03

Abstract

The United States Air Force and Department of Defense networks and information system are under attack from a variety of actors. Current network defense systems are reactive in nature and unable to prevent determined adversaries from successfully infiltrating these information systems. The realization of these facts led the Air Force Research Lab begin work on a next-generation network defense system called Cybercraft. The Cybercraft vision is a trusted, autonomous system which will perform network defense tasks.

In this paper, software engineering and threat analysis are used to create a set of initial requirements and system models for Cybercraft. This paper presents a methodology based on traditional software requirements elicitation processes and attack and defense trees to generate system requirements. Once requirements have been defined, they are used to create system use cases and a system domain model. This iterative process can be used to define the system in enough detail that software or system prototypes can be developed. The contribution of this paper is a set of initial requirements, use cases, and domain models which could be used in Cybercraft development. Ultimately, it is a generic methodology which could be used to determine requirements for any security system and how to apply those requirements to begin high-level system design.

To my wife and daughter for your love and support

Acknowledgements

I would like to express my sincere thanks to all of my fellow classmates in the Cyber Warfare program. You helped make this year fun and memorable. I would also like to thank Lt Col Todd McDonald for his expertise and assistance in creating this document. His guidance focused and improved this document immensely. I would like to thank Dr Bob Mills for his input which improved this work and also for his leadership in making the IDE Cyber Warfare program what it is today. Finally, I'd like to thank my wife and daughter for their understanding and patience through the year and especially as I worked to complete this paper. I love you both.

Michael G. Hunsberger

Table of Contents

List of Figures

AFIT/ICW/ENG/08-03

A METHODOLOGY FOR CYBERCRAFT REQUIREMENT DEFINITION AND INITIAL SYSTEM DESIGN

I. Introduction

The United States military relies on information systems for nearly every aspect of its operations. The advances in these systems have enabled the military to operate with amazing speed and precision. Because of the reliance on these systems, adversaries seek to deny, degrade, destroy and manipulate the data in them. Successful attacks on even a small subset of these systems would have far-reaching, negative effects on the military's ability to conduct operations. In his testimony before the House Armed Service Committee, General Cartwright, then Commander of the United States Strategic Command said these systems are also "under widespread attack… and our freedom to use cyberspace is threatened. We lack dominance in cyberspace and could grow increasingly vulnerable if we do not fundamentally change how we view this battle-space." (Cartwright, 2007)

The military's network defense systems and architectures are incapable of preventing determined adversaries from infiltrating these critical systems. Current defense systems are reactive in nature and cannot keep pace with continuously changing threats. (Goldman & Woodward, 2008) In its report, The Implications of Cyber Warfare, the Air Force Scientific Advisory Board provided recommends to mitigate some

of the effects of the current threat environment. Their first recommendation was to "Continue investment in technologies that attempt to maintain the integrity of networks and computer systems." (United States Air Force Scientific Advisory Board, 2007)

It is because of this threat environment and the inadequacy of current network defenses that the Air Force Research Lab (AFRL) began working on a revolutionary project to define the next-generation network defense capability. The project, called Cybercraft, seeks to describe the interface specification required for a *trusted*[1], autonomous system which will work with other Cybercraft to protect the Air Force's networks. The proposed Cybercraft architecture will consist of a long-life hardware platform which will implement payloads to execute network defense tasks. These tasks could include intrusion detection, anti-virus, host integrity, or firewall services. The platform will provide enough flexibility that it could run payload modules which have not been conceived of at this time. The goal of the proposed Cybercraft project is an architecture that could scale to one million nodes across the Air Force enterprise.

1.1 Research Motivation

Cybercraft project development has concentrated on the system architecture design without much emphasis on system requirements or scope definition. Cybercraft's ability to meet future defense needs will depend extensively on the requirements upon which the system is based. This research seeks to define a requirements development methodology for security systems which can be used to define the initial system requirements for Cybercraft.

[1] Emphasis added

1.2 Research Contribution

This document provides an iterative requirements development methodology for security systems using attack and defense tree modeling, use case definitions, and domain modeling. Attack and defense trees provide a process to decompose security threats and defenses by defining an attacker's goals, what steps they might take to achieve them, and what steps a defender might use to stop them. A use case is a text description of an actor's interaction with a system with a goal of defining what service the system is providing for the actor. Finally, domain modeling seeks to provide a visualization of the system in its operating environment through various levels of fidelity. This methodology was applied to current and emerging threat environment in which AFRL's Cybercraft project will operate in an effort to further the Cybercraft development effort.

1.3 Thesis Organization

This document is divided into five chapters. Chapter II presents background information on the individual pieces of the methodology. Chapter III describes the requirements development methodology in detail. It examines how user requirements and a vision and scope document fit into the methodology as well as discussing attack and defense trees, use cases, and domain modeling. Chapter IV provides the results and analysis of those results obtained by applying this methodology to the Cybercraft project. Chapter V summarizes the document and provides several recommendations for future work.

***Problem Statement*:** This research provides a structured, iterative methodology for security system requirements definition and initial system modeling. This methodology includes using attack and defense trees, use cases and domain modeling to define requirements and provide an initial visual system and environment description.

II. Related Work

Developing any new software or software-intensive system is a complex and challenging process. The field of software engineering has emerged to provide processes and procedures that facilitate complex software development. Software Development Life Cycle (SDLC) has emerged as an umbrella term to reference the steps or processes used to develop software.

The first step in nearly all models representing the SDLC is requirements elicitation and analysis. Defining requirements is critically important because it is the foundation on which the rest of the system will be built. According to Wiegers, Cosmic Truth #1 of software requirements is "If you don't get the requirements right, it doesn't matter how well you execute the rest of the project." (Wiegers, 2006)

Numerous techniques exist for identifying security threats to systems and software. These can also be applied within the context of Cybercraft to determine what capabilities the Cybercraft system should provide to Air Force network defenders.

This chapter will introduce the software engineering concepts of SDLC, requirements, use cases, and domain modeling. It will also introduce the security modeling techniques of attack trees and protection trees. Finally, the Cybercraft project will be introduced.

2.1 Software Development Life Cycle

Software development has evolved from the repetitive steps of code, test, and fix methodology to more rigorous process models that facilitate large-scale software product development. These process models are defined collectively as Software Development

Life Cycle (SDLC) models. While there are numerous models that fit under the SDLC definition, all are based on a procedural set of actions developers take when developing code. The waterfall model (Royce, 1987), spiral development model (Boehm, 1988; McCracken & Jackson, 1982), and the unified process (Jacobson, Booch, & Rumbaugh, 1998) are just a few of the SDLC models that have been proposed and used for software development. While discussion of individual models is beyond the scope of this document, Larman and Basili (Larman & Basili 2003) provide an excellent paper on the history of iterative and incremental development models which are used extensively today and will be applied in this development methodology.

2.2 Requirements

Requirements are the most important part of software development. They are also the most difficult. In his seminal paper, No Silver Bullet Essences and Accidents of Software Engineering, Fred Brooks described the challenges inherent in the requirements process as:

> The hardest single part of building a software system is deciding precisely what to build. No other part of the conceptual work is as difficult as establishing the detailed technical requirements, including all the interfaces to people, to machines, and to other software systems. No other part of the work so cripples the resulting system if done wrong. No other part is more difficult to rectify later. (Brooks, 1987)

Sommerville and Sawyer define requirements as a "specification of what should be implemented. They are descriptions of how the system should behave, or of a system property or attribute. They may be a constraint on the development process of the

system." (Sommerville, Sawyer, 1997) Requirements describe both "what" the system will do and also "how" it should behave.

Developers need these requirements to build the system. They must also realize that each stakeholder in the process will have a different view of requirements. To capture this, Wiegers introduces a process model (Figure 1) and four types of requirements which are functional and additional non-functional constraints or parameters. (Wiegers, 2003)

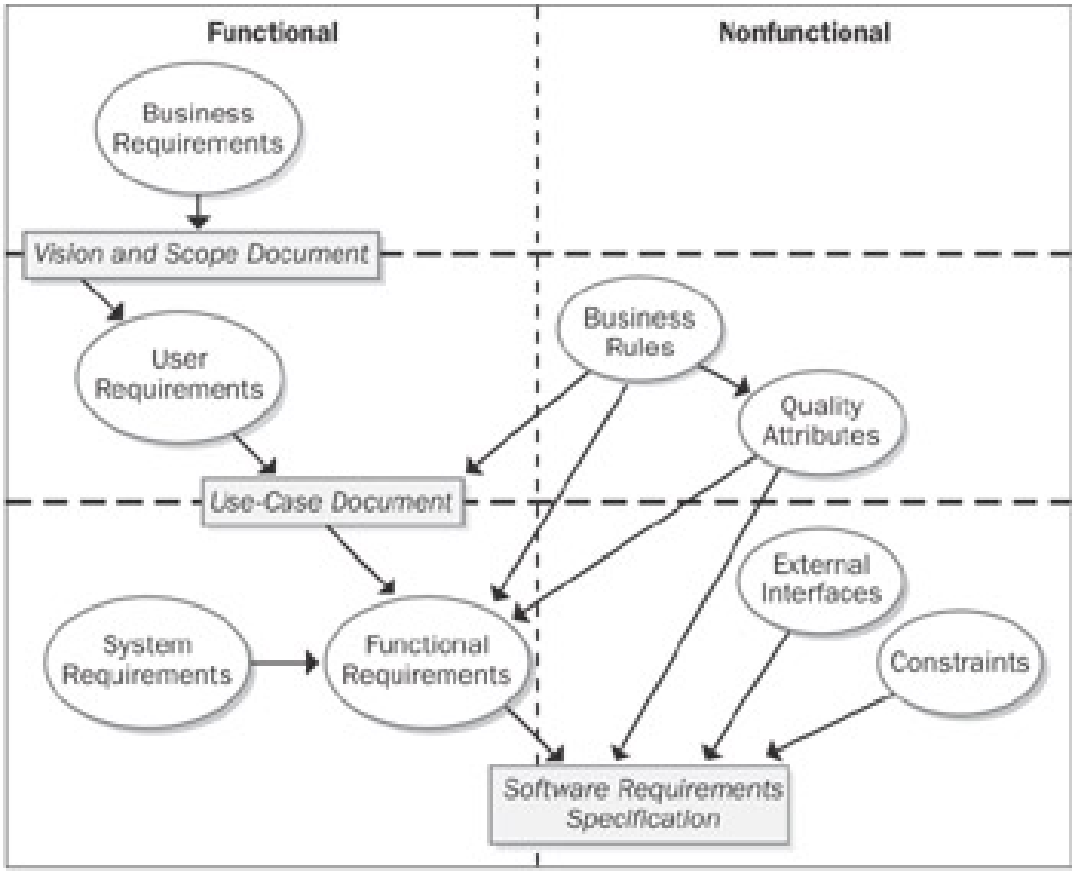

Figure 1. Relationship Of Several Types Of Requirement Information. (Wiegers, 2003)

Business requirements are the top-level requirements that define the system scope and objectives. Business requirements are also used to record the reason for creating the

system typically in a vision document or white paper. An example of a business requirement is: Develop a host-based intrusion detection system (IDS) for the Windows Vista Operating System.

User requirements are tasks or functions that the system user will be able to accomplish using the system. An example of a user requirement is: Update intrusion signatures from a centralized signature server.

Functional requirements specify to the developer the way the system needs to function. These are also known as behavior requirements. These are the "shall" statements in a software requirements specification (SRS). An example functional requirement: The system shall provide the user an alert if intrusion signatures are more than 30 days old. System requirements provide top-level requirements for sub-systems (software or hardware) with which the system under design may interface.

Non-functional requirements represent system constraints or attributes that do not contribute to the functionality of the system. These include business rules, quality attributes, external interfaces, and constraints. Business rules are policies or regulations which may affect who can use a system or how it can be used. In an IDS, this could be a business rule stating that only authorized technicians can change the intrusion signatures. Quality attributes are used to describe such things performance, security, or usability requirements. An IDS performance attribute could specify the system must process at least 100 megabits of data per second.

System and user interface information are provided in external interface specifications. Interface specifications could detail the graphical user interface used to enter data into a systems. Constraints are used to restrict some aspect of system design or

development. A development constraint could be the development language the developers need to use.

All of these requirements are typically represented in the SRS which documents "essential requirements (functions, performance, design constraints, and attributes) of the software and its external interfaces." (Institute for Electrical and Electronics Engineers (IEEE), 1991) This is used as the basis for requirements prioritization, grouping, management, and system development.

2.3 Use Cases

Use cases are a useful tool to capture an agreement between users and system developers about what a system needs to do. The use case "describes how an actor uses a system to achieve a goal and what the system does for the actor to achieve that goal. It tells the story of how the system and its actors collaborate to deliver something of value for at least one of the actors." (Bittner & Spence, 2003) An actor is someone or something which exhibits a behavior or performs an action. This could be a person, another system, or even a corporation. (Adolph & Bramble, 2003)

The goal is to create a set of use cases that fully specifies everything the user needs to accomplish when using the system. While this completeness is probably unrealistic it should provide the majority of scenarios from the point of view of each actor. (Wiegers, 2003)

Use cases are written in easy-to-read, text formats. Use cases can be categorized in numerous ways. Larman describes three use case categories of brief, casual, or fully dressed (also known as formal). (Larman, 2005) The brief use case is a simple, one page

summary of the actor's interaction with the system. Casual use cases are also written in unstructured paragraph format describing the actor and system interaction and what the system is doing for the actor. Fully dressed use cases provide additional information including pre- and post-conditions, the primary success scenario, and alternate scenarios that detail other success or failure paths. Fully dressed use cases are typically represented in table format.

Use case diagrams are another useful tool that can provide a context for the system. The diagram provides a way to view the system, system boundaries, the actors, their goals, and their interactions in a succinct, high-level view. Figure 2 shows an example use case diagram for a college course registration system. It details the functions each actor needs to accomplish by using the system.

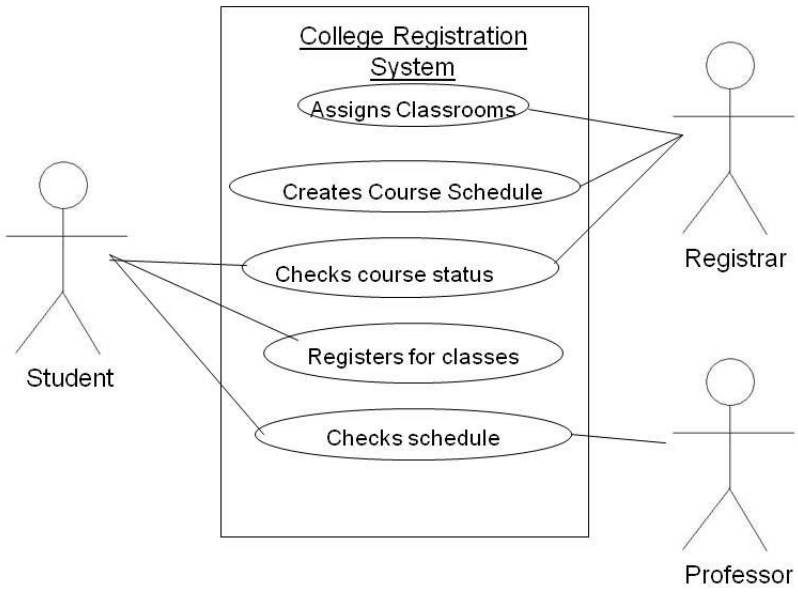

Figure 2. Notional Use Case Diagram

2.4 Domain Modeling

Within Object Oriented (OO) analysis, a key step is creating the domain model to represent the system. A domain model is a "conceptual perspective of objects in a real situation of the world, not a software perspective." (Larman, 2005) Domain models typically include the conceptual classes, the associations between these objects, and the attributes of the classes. Conceptual classes are used to define a thing or idea.

A notional example of a student registering for a class at a college will be used to illustrate the domain model concept. Student and Class are two conceptual classes shown in Figure 3. Student has attributes of the student's name, ID number, and address while class has attributes for the day or days the class meets, the time, and the number of credits the course is worth.

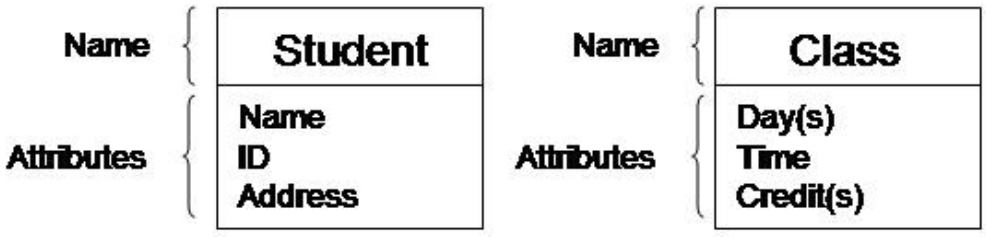

Figure 3. Notional Conceptual Classes

Figure 4 shows a notional, partial domain model with the classes in Figure 3 and additional classes linked by association. The Student class registers for Class. This is a many-to-many relationship because multiple students may take multiple classes. This relationship is represented in by the * on the association line. The Professor class teaches Class. This is a many to many relationship, but (in this example) a Class will have one, two, or three Professors assigned to the Class. The Class Is In association is also many-

to-many because a Class could be in a multiple rooms on different days and the Room will have multiple Classes.

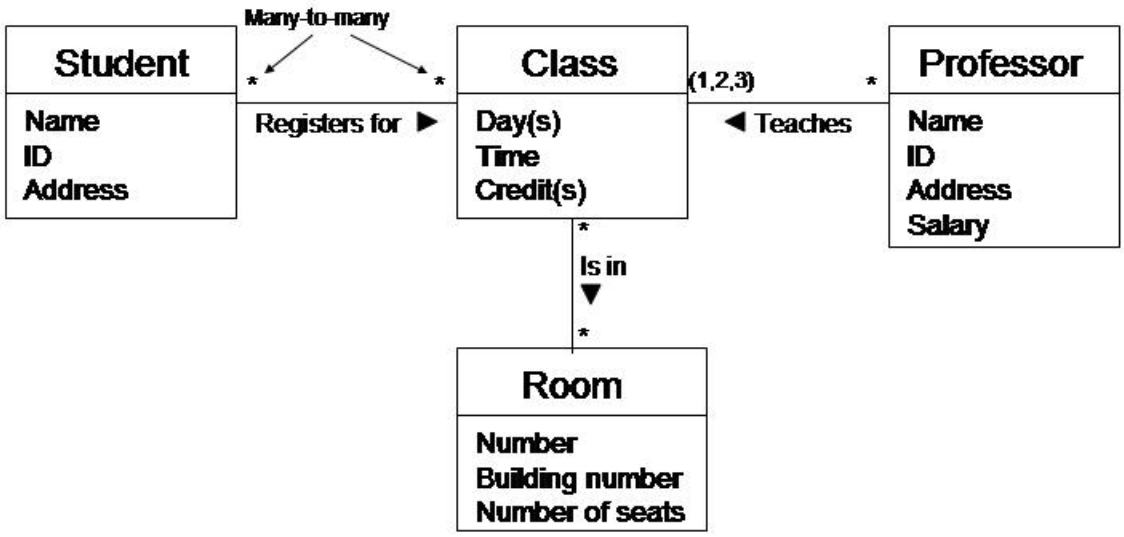

Figure 4. Partial Domain Model

The key to successful domain modeling is that it is an iterative process applied in conjunction with requirements elicitation. It should not be seen as a one-time exercise to define the complete domain model. As requirements evolve or new requirements are identified, the domain model will also change.

2.5 Attack Trees

Attack trees provide a methodical way to represent attacks that may be applied against a system. Attack trees "represent attacks against a system in a tree structure, with the goal as the root node and different ways of achieving that goal as leaf nodes. " (Schneier, 1999) The root goal of the attacker is the expressed as the root of the tree

which is then decomposed with various attack methods that might be used to achieve this goal. Attacks are further decomposed into sub-attacks. This is continued until attacks cannot be further decomposed which is represented as a leaf of the tree.

Figure 5 shows an example attack tree for preventing someone from using their car. The single lines from parent to child node indicates an OR statement. In this notional example, the attacker has four options to prevent someone from using their car. If the attacker successfully executes any of the four options, he will have achieved his goal. Lines from a parent to child node that have an arc connecting them indicate an AND relationship. In attempting to steal the victim's car, the attacker would need to break into the car AND then hotwire it to successfully steal the car.

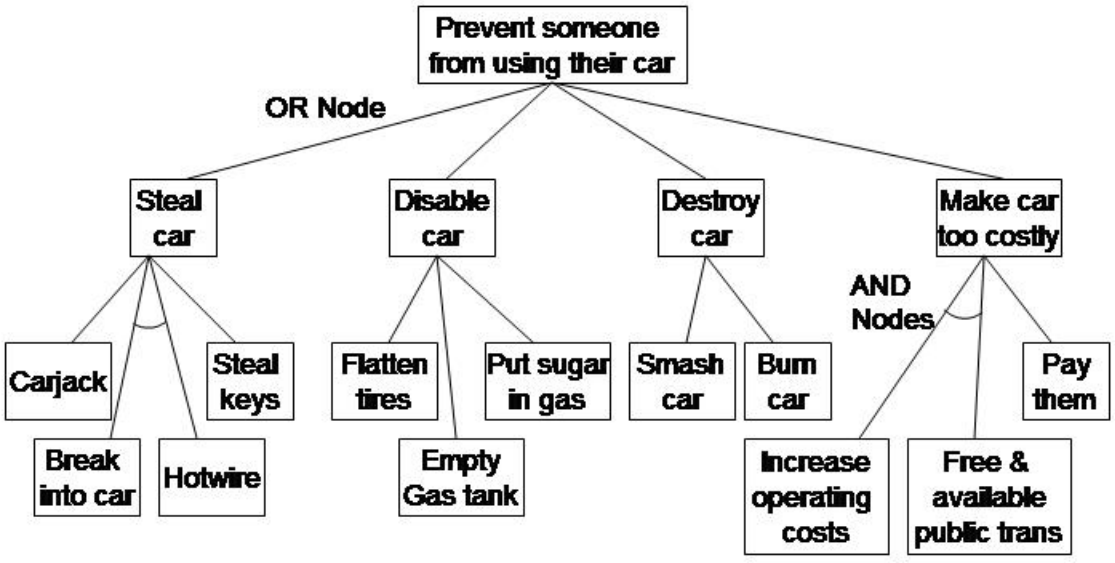

Figure 5. Example Attack Tree for Prevent Use of Car

Figure 6 represents another way to create an attack tree using a text format. All of the information is transferred including the use of AND and OR nodes. This format is useful for highly complex attack trees with multiple dependencies and child nodes.

```
GOAL:  Prevent someone from using their car
OR  1.0 Steal Car
        OR 1.1  Carjack
           1.2  Steal keys
        AND 1.3 Break into car
            1.4 Hotwire
    2.0 Disable car
        OR 2.1  Flatten tires
           2.2  Empty gas tank
           2.3  Put sugar in gas tank
    3.0 Destroy car
        OR 3.1 Smash car
           3.2 Burn car
    4.0 Make car to costly
         OR 4.1  Pay them not to drive
        AND  4.2  Increase operating costs
                  4.3  Free and available public transportation
```

Figure 6. Example Attack Tree in Text Format

Once all of the possible attacks which may be applied to reach the root goal have been enumerated, values can be assigned to each node. These values could be simple Boolean values of I (impossible) or P (possible), special tools required or no special tools required, or expensive or inexpensive. They could also take real values of cost to implement, probability of success, or time required. (Schneier, 1999)

These node values can be aggregated to the parent node to determine possible attacks, the cheapest attacks, attacks with the highest probability of success, or takes which could be completed the fastest. Node values may vary based on individual attacker profiles. The threat from a terrorist who is willing to die for their cause is likely

different from organized crime or state sponsored attackers that may have extensive resources available to carry out an attack. The combination of these attack trees will give the defender the best comprehensive view of the attack surface that must be defended.

2.6 Protection and Defense Trees

While attack trees provide the formalized way to illustrate vulnerabilities in a system, they do not provide discussion of mitigating actions. Protection or defense trees provide a methodology to prune or alter attack trees based on proposed or implemented defenses. This analysis can provide the defender the ability to determine the best defense mechanisms based on resources required to implement. (Edge, 2006)

Figure 7 shows the example attack tree with notional node values for both the probability of attack and estimated cost to execute the attack. This attack tree will be used to create a notional protection tree that will prune or alter the attack tree.

GOAL: Prevent someone from using their car	Probaility of Attack Success	Cost (in $K)
OR 1.0 Steal Car		
OR 1.1 Carjack	.70	1
1.2 Steal keys	.35	0.5
AND 1.3 Break into car	.80	0.5
1.4 Hotwire	.50	2
2.0 Disable car		
OR 2.1 Flatten tires	.95	0
2.2 Empty gas tank	.60	0.25
2.3 Put sugar in gas tank	.45	0.1
3.0 Destroy car		
OR 3.1 Smash car	.35	5
3.2 Burn car	.5	1
4.0 Make car to costly		
OR 4.1 Pay them not to drive	.5	10
AND 4.2 Increase operating costs	.25	100
4.3 Free and available public transportation	.25	100

Figure 7. Example Attack Tree with Node Values

The notional protection tree in Figure 8 shows some of the things which could be done to allow the victim freedom to use their car. Since most defenders are limited by the amount of resources they can use to protect against attacks, they must prioritize what protection measures are applied. For this example protection 1.3, hire 24/7 security for the car, will be applied to the notional attack tree. Figure 9 shows the modified attack after the protection tree has been applied.

GOAL: Provide freedom to use car	Cost (in $K)
1.0 Only park where car is protected	
1.1 Park in garage	10
1.2 Park in monitored locations	5
1.3 Hire 24/7 security for the car	50
2.0 Carry a gun for protection	
2.1 Buy gun	1
2.2 Register gun	0.1
3.0 Use locking gas cap	0.01
4.0 Have enough money to operate the car	
4.1 Get a better job	2
4.2 Have someone give you money	0
4.3 Steal money	1

Figure 8. Example Protection Tree

GOAL: Prevent someone from using their car	Probaility of Attack Success	Cost (in $K)
OR 1.0 Steal Car		
OR 1.1 Carjack	.10	20
1.2 Steal keys	.10	15
AND 1.3 Break into car	.10	15
1.4 Hotwire	.10	2
2.0 Disable car		
OR 2.1 Flatten tires	.05	10
2.2 Empty gas tank	.05	10
2.3 Put sugar in gas tank	.05	10
3.0 Destroy car		
OR 3.1 Smash car	.15	15
3.2 Burn car	.05	15
4.0 Make car to costly		
OR 4.1 Pay them not to drive	.5	10
AND 4.2 Increase operating costs	.25	100
4.3 Free and available public transportation	.25	100

Figure 9. Revised Attack Tree after Protection Measures Applied

2.7 Cybercraft

Cybercraft is an active Air Force Research Lab (AFRL) project creating the interface standards specification for the next-generation cyber defense tool. This program was initially proposed by Phister, et al. (Phister, Fayette, & Krzysiak, 2005) as a cyber vehicle operating only in the cyber domain. It could:

> "perform similar operations as conventional vehicles (e.g., UAV's), such as a "strike" platform (e.g., deny, destroy, degrade, disrupt or deceive) or as an "ISR" platform (e.g., find, fix, track, monitor). The characteristics of a "Cyber-Craft" include the ability to be launched from a network platform, the ability to embed control instructions within the craft, the ability to positively control the "Cyber-Craft" from a remote network location, the capability for the craft to self-destruct upon being recognized, the capability for the craft to operate with minimal or no signature/footprint, and the ability for the "Cyber-Craft" to rendezvous and cooperate with other friendly "Cyber-Craft". [2]

[2] Henceforth referred to as Cybercraft

The current AFRL project has defined Cybercraft as "a trusted computer entity designed to cooperate with other Cybercraft to defend Air Force networks." (Bibighaus, 2006) and "… a trusted platform for automated C3 [Command, Control, Communications] and delivery of defensive cyber capabilities." (Glumich, 2008) The system, based on a secure design, will perform autonomous real-time cyber defense actions. The autonomous characteristics of Cybercraft introduce the issue of trust within the system. The fundamental research question has become how to design a system that commanders and operators trust to do what it is supposed to do and nothing else.

An additional requirement is that only authorized operators may control the system. The Cybercraft operators should also be the only users that can alter system configurations. Users, even those with root or administrator privileges on the system or domain must be unable to disable or alter Cybercraft configurations unless they are also authorized Cybercraft administrators.

Cybercraft development has focused on six fundamental research areas. They are: Map and mission context; Environmental Description, C3 Protocols and architecture; Formal model and policy; Self-protection guarantee; Interfaces and payload. AFRL has formed working groups that are examining each of these areas and have executed several contract efforts aimed at furthering this research. In addition to the six research areas, there is a requirements working group defining the system's functional objectives for the six research areas.

The proposed Cybercraft architecture consists of a Cybercraft platform, payloads, which will be executed by the platform, and associated control infrastructure. The payload is a long service life, trusted, hardware or hardware and software device. It will

execute Cybercraft payloads to perform numerous types of missions. The Cybercraft payloads will be expendable code which can be rapidly developed to perform a cyber defense mission.

2.8 Summary

The purpose of this research is to identify a structured, repeatable process to which can be used to define Cybercraft requirements and create the Cybercraft interface specification standard. This chapter presented some of the related concepts that will be used to define Cybercraft requirements. This included a discussion of the software development concepts of the SDLC, requirements, use cases and domain modeling. It also presented background security methodologies attack and defense trees which will be used to enumerate Cybercraft requirements. Finally, the AFRL Cybercraft project, its origins, and goals were presented. The next chapter will present the methodology used to define Cybercraft requirements while chapter IV will present the initial results using this methodology.

III. Methodology

The creation of non-trivial software or a software specification, in the case of Cybercraft, requires a repeatable requirement definition methodology. This chapter presents a proposed methodology which if followed will provide Cybercraft with a useful set of requirements for use in the Cybercraft design. The candidate methodology is a blend of traditional requirements elicitation methodology and agile and unified process concepts.

The process begins with an examination of the scope and vision for the overall system. Next user requirements are gathered in parallel with system and security experts creating attack and protection trees. The elicited requirements should then be prioritized based on user requirements and development constraints. Finally, an iterative process of using the defined requirements to create use cases and create and refine system domain models will be used to provide additional system detail.

The chapter begins with discussion of the vision and scope document. Section 3.2 details the requirements elicitation process while 3.3 provides information on user requirements. Section 3.4 and 3.5 discuss attack and protection trees. Finally section 3.6 and 3.7 discuss the iterative process of use case and domain model creation and refinement. Section 3.8 provides the chapter summary.

3.1 Vision and Scope Document

The first step in any system development should be to define the overall requirement and business case for the system. It needs to identify numerous things including:

What will the system do?
To whom will the system provide services?
What is the need for the system?
What is the justification for the system?
What system(s), if any does the system replace?

The vision piece of the document ensures all interested parties agree on what the system will do. The scope piece defines the system boundaries including what will and will not be included in the system. This project scope should be broken down into the individual releases with each iteration having its own scoping document. This keeps the development focused and allows the iterations to be easily aligned ensuring the overall project scope is achieved.

The scope and vision document should also contain the stakeholder profiles, project priorities, and the operating environment (Wiegers, 2003). Smith defines a stakeholder as "individuals, groups, or organizations that have an interest in the project and can mobilize resources to affect its outcome in some way." (Smith, 2000) The stakeholder profiles categorize the primary individual and group stakeholders and the expected benefits the new system will provide them.

Project priorities detail the order in which requirements should be completed. They require the stakeholders to agree on the organization, deliverables, and timing for the project. Prioritized requirements also identify the project's constraints and

tradespace. The operating environment describes the system within its operating context. It should define the overall system scope, and security aspects in terms of availability, integrity, and reliability.

3.2 Requirements Elicitation Overview

Functional requirements elicitation is critical to system design and operation. Using the wrong requirements will result in a system that does not meet the user's requirements without significant additional rework. Requirements elicitation is best completed as early in the SDLC process as possible. The later a requirements mismatch is found, the more expensive the resulting fix will be (as shown in Figure 10).

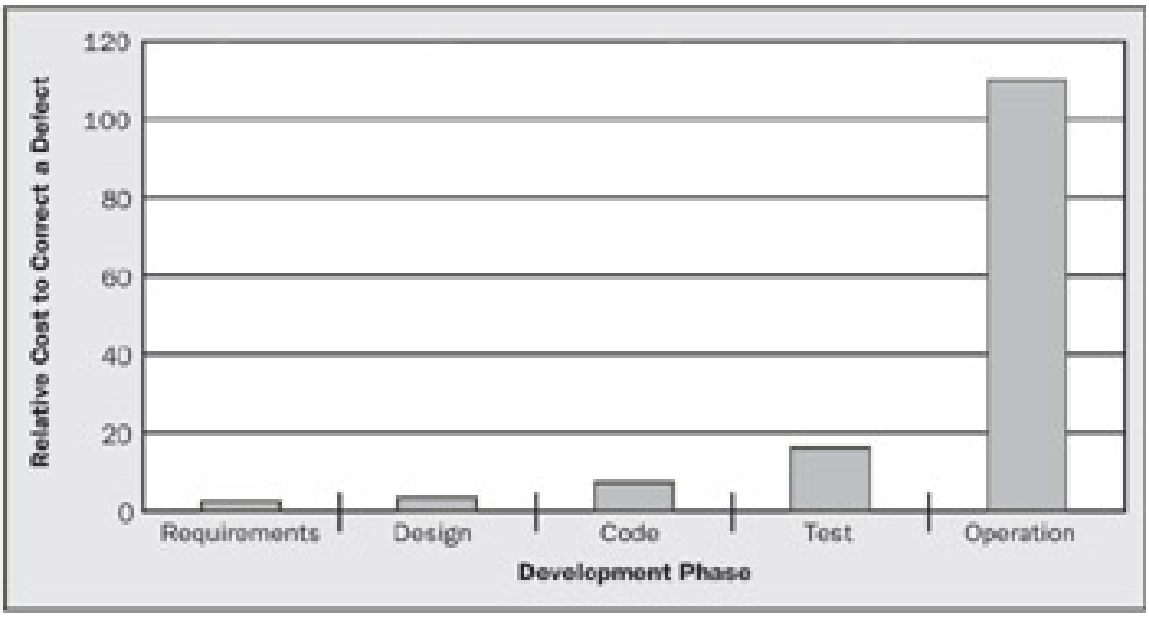

Figure 10. Relative Cost to Correct an Error across the SDLC (Wiegers, 2003)

User input and threat based analysis should be the two primary ways to define the requirements for Cybercraft. While user requirements may provide the bulk of the requirements for the system, it is also useful to look at the current and emerging environment from a threat perspective. This helps ensure nothing is missed from the larger, emerging threat environment. The next sections detail eliciting user requirements and the use of attack and defense trees for overarching threat issues.

3.3 User Requirements

User requirements are the most important part of the system requirement definition. Users will eventually operate the system. The system's ability to meet user needs will either result in success or require costly rework for the system as shown in Figure 10. It is much cheaper to gather the requirements early in the project than it is after the product has been delivered.

Systems usually have several distinct user classes which are based on how each class uses the system. In a typical web-based order system the classes might be end user, billing, order fulfillment, and shipping. Each of these users interacts with the system in a different way and will have different requirements for the system.

While user classes will typically have numerous members, it is often useful to pick a single or small number of users in a particular user class that can act as product champions. Product champions represent the entire user class and are empowered to work with developers and make decisions for the user class. Product champions are typically user class members with extensive experience within the product domain. They

act as the information conduit between the users and the developers by providing feedback and clarification and defining or conveying requirements.

The product champions should work closely with developers through the process, including project scoping, requirements development workshops, requirements prioritization, prototyping, and testing, for each project iteration.

3.4 Attack Trees

Another useful method for generating system security requirements is to investigate the threat environment in which the system will be operating. Attack trees (Mauw & Oostdijk, 2006; Schneier, 1999; Schneier, 2000) and threat modeling (Myagmar, Lee, & Yurcik, 2005; Swiderski & Snyder, 2004; Torr, 2005) were considered as candidate procedures for security requirement definition. Attack trees were chosen because they provide a structured, formal methodology for decomposing attacks. Attack trees also provide a modular framework which will facilitate distributed or parallel attack tree creation.

Attack trees are constructed by first defining an attacker's goal or goals for attacking the system. Each goal forms a separate attack tree. Once the goals have been enumerated, the attacks which might be used to achieve the goal are identified. This decomposition of attacks into sub-attacks is repeated until the sub-attacks cannot be further decomposed. These are represented as the leaves on the attack tree.

Attack tree generation requires extensive familiarity with operating environment, current system, and current and emerging threats. Additionally, a security mindset is extremely useful. For most people, the ability to think like an attacker or criminal is not

normal although it is required to excel at determining most security problems. (Schneier, 2008)

Additional resources are available to facilitate attack tree generation. Attack patterns are particularly useful because they generalize attacks that can be instantiated to provide defined attacks or sub-attacks. The US Department of Homeland Security through its National Cyber Security Division is sponsoring two initiatives which are useful in attack tree generation.

The first initiative is the Build Security In website which provides information and links to best practices, guidelines, and resources for secure software development across the Software Development Life Cycle. (Department of Homeland Security National Cyber Security Division, 2008a) The second initiative is the Common Attack Pattern Enumeration and Classification (CAPEC) website. This website contains information about attack patterns and an attack pattern library for developers use in designing more secure software. (Department of Homeland Security National Cyber Security Division, 2008b) While the attack dictionary is sparsely populated at this time, the goal is for the software development community will provide additional attack patterns which will increase the site's utility. Figure 11 shows part of an example attack pattern found in the attack pattern dictionary.

BACK TO TOP

Individual CAPEC Dictionary Definition (Release 1.1)

Buffer Overflow via Symbolic Links

Attack Pattern ID	45	**Pattern Abstraction: Detailed**
Typical Severity	High	
Description	**Summary** This type of attack leverages the use of symbolic links to cause buffer overflows. An attacker can try to create or manipulate a symbolic link file such that its contents result in out of bounds data. When the target software processes the symbolic link file, it could potentially overflow internal buffers with insufficient bounds checking. **Attack Execution Flow** 1. 1- The attacker creates or modifies a symbolic link pointing to a resources (e.g., file, directory). The content of the symbolic link file includes out-of-bounds (e.g. excessive length) data. 2. 2- The target host consumes the data pointed to by the symbolic link file. The target host may either intentionally expect to read a symbolic link or it may be fooled by the replacement of the original resource and read the attacker's symbolic link. 3. 3- While consuming the data, the target host does not check for buffer boundary which can lead to a buffer overflow. If the content of the data is controlled by the attacker, this is an avenue for remote code execution.	
Attack Prerequisites	The attacker can create symbolic link on the target host. The target host does not perform correct boundary checking while consuming data from a ressources.	
Typical Likelihood of Exploit	High	
Methods of Attack	• Injection • Modification of Resources	
Examples-Instances	**Description** Attack Example: Overflow with Symbolic Links in EFTP Server The EFTP server has a buffer overflow that can be exploited if an attacker uploads a .lnk (link) file that contains more than 1,744 bytes. This is a classic example of an indirect buffer overflow. First the attacker uploads some content (the link file) and then the attacker causes the client consuming the data to be exploited. In this example, the ls command is exploited to compromise the server software.	
Attacker Skill or Knowledge Required	Low : An attacker can simply overflow a buffer by inserting a long string into an attacker-modifiable injection vector. The result can be a DoS. High : Exploiting a buffer overflow to inject malicious code into the stack of a software system or even the heap can require a higher skill level.	

Figure 11. Partial Attack Pattern Example from CAPEC Dictionary (Department of Homeland Security National Cyber Security Division, 2008b)

There are several challenges when trying to create comprehensive attack trees. Complete attack trees can be quite extensive even for small systems. They can become truly unwieldy for large systems. Attack trees are also subject to the limitations of the tree designer's understanding of the systems and possible attacks. Additionally, comprehensive, multi-stage attacks are difficult to time sequence using attack trees although framework extensions have been proposed to facilitate these activities. (Daley, 2002) All of these limitations can lead to analysis errors when using the attack tree methodology. (Edge, 2006)

Once the attack trees have been designed, node values need to be assigned to the tree's leaves. This facilitates the defense mechanism prioritization. Nodal values can take numerous forms. They can vary from simple binary values of possible or impossible and expensive or inexpensive to probability of attack occurring, probability of attack success, impact to the system, or cost in equipment or time required to execute the attack. The overall risk to the system is the most useful metric for attack tree nodes.

$$risk = (probability/cost) \times impact$$

Equation: Risk Calculation for Leaf Nodes (Edge, 2006)

These node values are repeatedly propagated up to parent nodes to determine the overall risk to the system for the root attack goal.

3.5 Protection Trees

Once attack trees have been fully enumerated, it is useful to think about how to eliminate or at a minimum mitigate the attacks that might occur on the system. This can be accomplished by creating protection trees (Edge, 2006) which will detail implemented or planned security mechanisms.

The first step in protection tree design should be to create a tree based on the existing system protections. This prevents duplication of effort and the application of protections that may not further affect the attack tree. Once the current system protection tree is designed, the next step is to perform a node analysis of the initial attack tree. This

is done by modifying the node values or pruning the leaves based on the defenses provided in the protection tree. The protection tree may change the node values such as cost, likelihood of success, or probability of the attack which will affect the overall risk of that root attack goal.

The new attack tree (AT Revised) is the remaining residual threat environment for the system. If resources remain that can be applied to mitigate the risks, another protection tree should be created with planned defenses. The new protection tree should again be applied against the residual attack tree to create the revised residual attack tree (AT Rev 1). This process could be repeated in serially as shown in Figure 12. The process could also be applied in parallel. Parallel creation (Figure 13) will provide a mechanism for tradespace analysis of potential defensive measures. This process should continue until the risk from the threat environment is determined to be acceptable or defense resources have been exhausted.

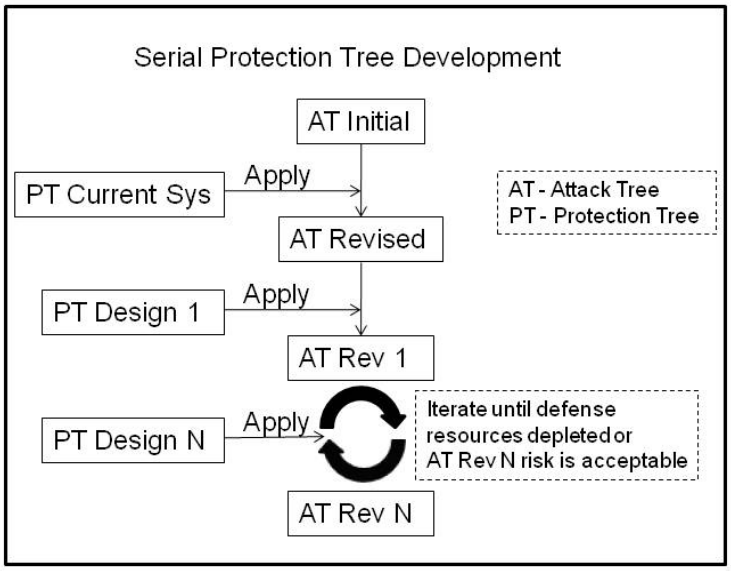

Figure 12. Serial Protection Tree Development

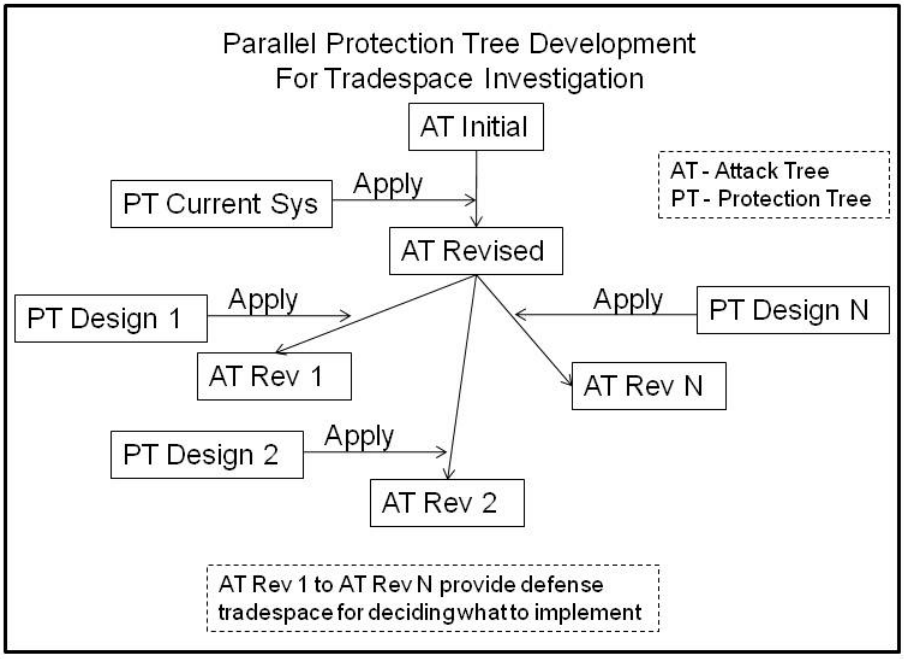

Figure 13. Parallel Protection Tree Development for Tradespace Analysis

3.6 Use Case Definition

Once initial user requirements and protection trees have been defined, we can start to build use cases that will provide additional needed for system design. A use case is a method for detailing system requirements by identifying an actor and the task they will accomplish using the system. Use cases seek to provide an understanding between the user and the development team and are best at detailing functional requirements. (Bittner & Spence, 2003; Cockburn, 2001)

Use cases are typically written as text documents that can easily be understood by both users and developers. There are numerous use case types (terse, informal, casual, brief, formal, and fully dressed) which have varying levels of detail and format variations. Terse, brief, and fully dressed will be used in this methodology.

Terse use cases typically contain one or possibly several sentences which convey at the highest level the summary of what the actor is trying to accomplish. Brief use cases are written in paragraph format and provide the main step by step success scenario for the action. Finally, the fully dressed use case is a formalized, template-based use case with preconditions, the main success scenario, alternate and error conditions, and the final success condition.

The first step in use case development is to take the existing requirements and protection tree scenarios and convert them into terse use cases. This is considered the "mile wide, inch deep" (Larman, 2005) approach. Once this is completed, the use cases should be prioritized by the project champions and development team (figure 14). Based on the prioritization, the terse use cases should be expanded into brief and fully dressed formats. This is required to the developers the additional information necessary to create design information.

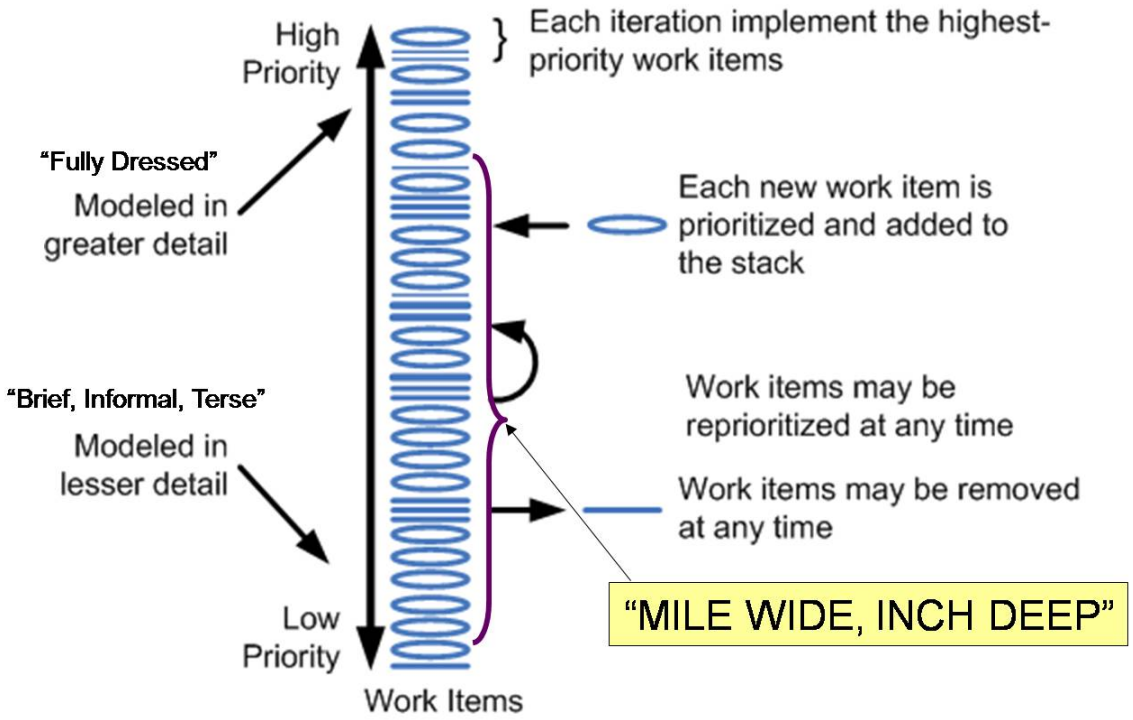

Figure 14. Use Case Strategy (McDonald, 2008)

3.7 Domain Model Creation and Refinement

In conjunction with use case creation, developers should create the domain model representation. The domain models serves as a "visual dictionary" and represents real-world objects in the operating environment. (Larman, 2005) Use case development and domain modeling is an iterative process aimed at increasing system design specificity to provide a visual context for the system.

The first step in creating the domain model is to determine the conceptual classes which will be part of the system. These are the real world objects that are part of the domain. Objects can be found in the vision and scope document, initial requirements, use cases, and expert knowledge of the problem space. (Rosenberg & Scott, 1999) Within

the use cases, the nouns used are typically good candidate for conceptual classes. Additionally, other models within the same problem space might have classes that can also be used.

The next step is to draw the initial class diagram. At this point the developer needs to decide on the abstraction level necessary to portray the system. The level-0 class diagram contains the highest level objects which will be further decomposed with additional details in subsequent domain models.

The third step is to create the associations between the conceptual classes. This step is used to determine what the links are between objects. The primary source for finding an association is the verbs found in use cases and the common real-world association between objects. The domain model diagram should be updated at this point to reflect the associations between the objects.

Finally, we add object attributes to the domain model. Attributes are the descriptive information about the objects that will be represented in the model. These can be found in the verbs and adjectives in use cases. Attributes are typically represented as string or number data types to provide additional information about the object.

The domain modeling process is iterative and evolutionary. Extensive time should not be spent on the initial effort because it will change as additional use cases are modeled and additional system requirements are added.

3.8 Summary

This chapter discussed the methodology which could be used to define requirements needed to create the Cybercraft software specification. This chapter

discussed the traditional requirements process aspects of the high-level scope and vision document and user requirements elicitation. It also presented the attack and protection process for security requirement definition. Next, the agile and unified process methods of iterative use case and domain modeling creation and refinement. Chapter IV provides analysis and results from the application of this methodology to define Cybercraft requirements.

IV. RESULTS AND ANALYSIS

This chapter provides and discusses the results of the proposed development methodology, described in the previous chapter, in context of the Air Force Research Lab's Cybercraft project. Section 4.1 examines the system's purpose and the need for an overarching project scope. The next section discusses user requirements while sections 4.3 and 4.4 describe how attack and defense trees can be used to define requirements. Section 4.5 discusses how use cases can provide additional details from high-level requirements and how the use cases can be used to define the domain model at various levels of fidelity. Finally, section 4.6 provides a chapter summary.

4.1 System Purpose

The first thing a project team must do when initiating a new system is to answer the critical questions of what will the system do, for whom will the system provide service, and why is the system needed. This ensures all stakeholders have a clear understanding of both the problem to be solved and what will and will not be included in the system. The scope and vision document should record this information. This is "especially critical for multisite development projects" like Cybercraft, "where geographical separation inhibits the day-to-day interactions that facilitate teamwork." (Wiegers, 2003)

Although it falls short in several respects, the Bibighaus Cybercraft whitepaper (Bibighaus, 2006) is the closest Cybercraft product to a scope and vision document. It defines the high-level vision for the Cybercraft project. The two overarching goals are to define the interface standards required to build Cybercraft and that Cybercraft will be a

trusted cyberspace vehicle that works with other Cybercraft to defend the Air Force network. While neither the project team nor AFRL will be the ultimate production facility, the team intends to produce a 1,000 node prototype to test design tradeoffs.

The whitepaper falls short in that it does not adequately scope the Cybercraft project. There are numerous systems currently in place that defend the Air Force network. The whitepaper and subsequent working groups have not specified whether Cybercraft is intended to augment or replace some or all of the current and planned network defenses. The whitepaper also details some of the intended Cybercraft architecture. It should only present the Cybercraft operating environment and any high-level architecture constraints. The actual architecture decisions should be presented in project design documents once initial requirements and domain models have been created and refined.

While the Cybercraft whitepaper was not intended to be a fully detailed scope and vision document, it is the closest product available for the project. Cybercraft project leadership should either modify the whitepaper or create a scope and vision document which can provide this information to the development team and Cybercraft stakeholders to better facilitate on-going development efforts.

4.2 User Requirements

Air Force network defenders, who will be the ultimate end users of Cybercraft, have had little involvement in the system requirement specification process. The project has considered the top ten network defense priorities from the 2006 Air Combat Command Information Operations Requirements and Architecture Working Group

(McDonald, 2007). Hunt also provided several possible requirements based on work with the Network Defense Lead from Detachment 3 of the 83rd Network Operations Squadron, but these only identify a small subset of probable requirements for Cybercraft. (Hunt, 2008)

Cybercraft developers should engage with Air Force network defenders and planners across all levels of the current Air Force defense architecture from base-level to the Integrated Network Operations and Security Centers (I-NOSC) and the I-NOSC detachments and finally to the Air Force Network Operations and Security Center (AFNOSC). Cybercraft project leaders should also engage with the planners and architects who understand the Air Force network operations and security architecture of tomorrow and the impacts it will have on the Cybercraft requirements and architecture.

This will provide Cybercraft project leadership with an invaluable perspective on user needs and the expectations for a comprehensive network defense system. While this may not provide all of the Cybercraft requirements, it will further the process and assist with future operator buy-in. It will also prevent costly rework by giving the design team perspectives which may not have considered.

4.3 Attack Trees

Attack trees provide a methodical way of decomposing and visualizing network attacks against a system. Several notional attack trees were developed for decomposition and requirements gathering. The top level attack goals used in this example are:

1.0 Deny Use of Host

2.0 Degrade Use of Host
3.0 Compromise Host
4.0 Alter Host Data
5.0 Exfiltrate Host Data
6.0 Deny Use of Network
7.0 Degrade Use of Network
8.0 Compromise Network Node
9.0 Alter Network Data
10.0 Exfiltrate Network Data

Some of these were partially decomposed (Appendix A) into attack trees. The attack trees which can be applied against a host (Figure 15) were used for further analysis.

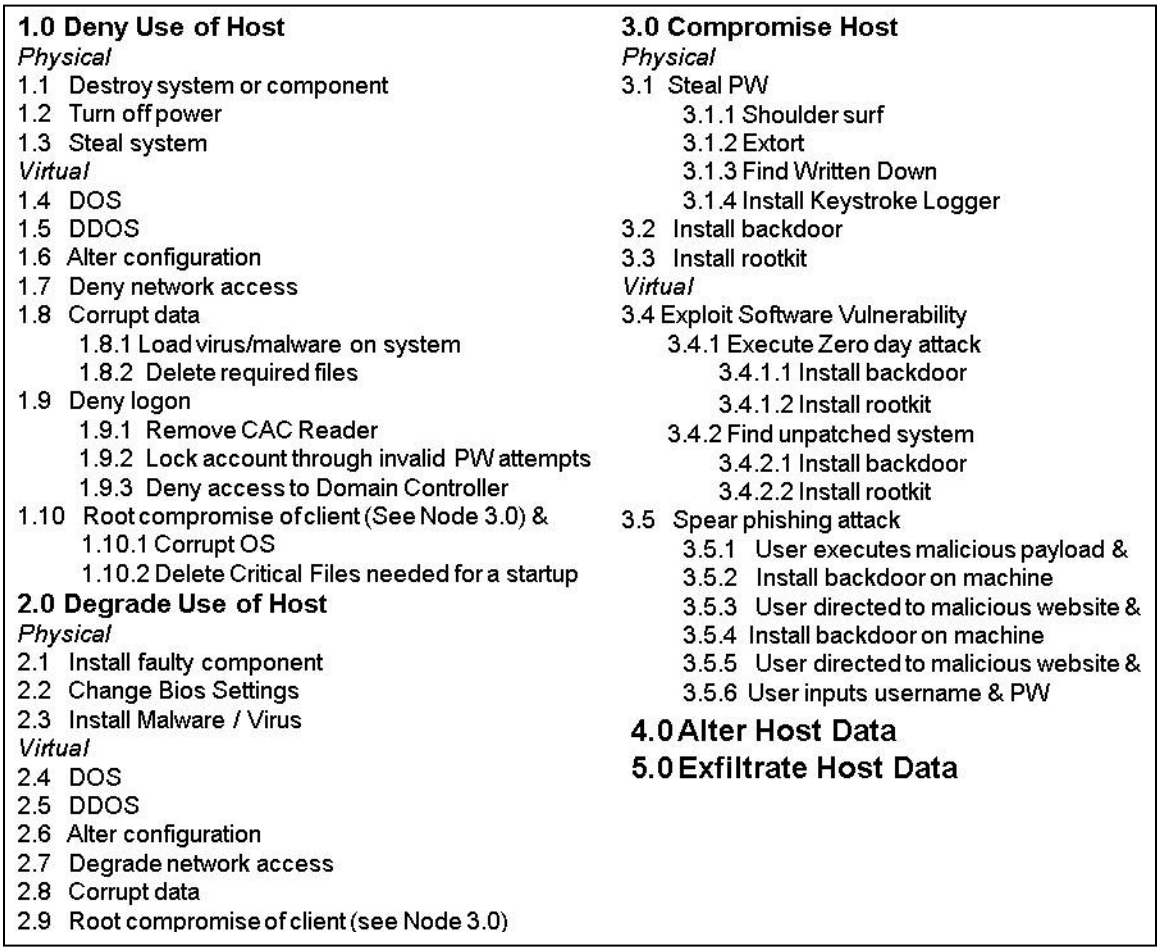

Figure 15. Attack Trees for Executing Attacks Against a Host

These attack trees highlight the fact that this is a multi-domain problem. While Cybercraft and technical solutions can provide extensive protections for our systems, there are numerous physical and user attacks that will still need to be mitigated.

Figure 16 shows an attack tree with notional risk values assigned. If representative values for attack trees can be gathered or estimated with a high degree of confidence, it will allow defenders and system developers to concentrate efforts on protecting against the highest risk attacks or those which might have the biggest impact on operations. Analysis of this notional attack tree shows host systems are most at risk from not being patched, having a rootkit installed, or falling victim to a spear phishing attack. The defense efforts should focus on mitigating these attacks. It should be noted that this type of analysis is only as good as accuracy of the numbers supplied. Basing protection development on faulty risk analysis may result in wasted resources and protecting the assets which may not be at risk.

		Probablilty	Cost	Impact	Risk
3	**Compromise Host**				
Physical					
3.1	Steal Password				
3.1.1	Shoulder surf	5	100	6	0.30
3.1.2	Extort	20	5000	6	0.02
3.1.3	Find Written Down	5	100	6	0.30
3.1.4	Install Keystroke Logger	10	200	6	0.30
3.2	Install Backdoor	25	200	8	1.00
3.3	Install Rootkit	40	200	10	2.00
Virtual					
3.4	Exploit Software Vulnerability				
3.4.1	Execute Zero Day Attack	5	5000	9	0.01
3.4.1.1	Install backdoor	20	500	8	0.32
3.4.1.2	Install rootkit	40	500	10	0.80
3.4.2	Find Unpatched System	70	200	5	1.75
3.4.2.1	Install backdoor	25	500	8	0.40
3.4.2.2	Install rootkit	40	500	10	0.80
3.5	Spear phishing attack				
3.5.1	User executes malicious payload &	35	500	8	0.56
3.5.2	Install backdoor	10	500	8	0.16
3.5.3	User directed to malicious website &	50	500	8	0.80
3.5.4	Install backdoor on machine	20	500	8	0.32
3.5.5	User directed to malicious website &	50	500	8	0.80
3.5.6	User inputs username & password	25	500	8	0.40

Figure 16. Attack Tree with Notional Values

4.4 Defense Trees

The next step in this requirement elicitation methodology is to develop defense trees to mitigate attacks detailed in the attack trees. The defense trees which provide useful mitigation can be considered as a candidate requirement for the Cybercraft system. Because this methodology is being use to elicit Cybercraft requirements, the defense trees developed are primarily focused on technical solutions. It should be noted that physical and user solutions may also be needed to successfully defend against attacks.

The root node of a defense tree is created to match the root node of the attack tree. Figure 17 shows a defense tree created to mitigate the effects of attack tree 3.0 Compromise Host (Figure 15). The root goal of the defense tree is to prevent host

compromise. The defense tree shows a subset of the numerous available techniques which could be used for mitigation. Both physical and virtual measures are shown.

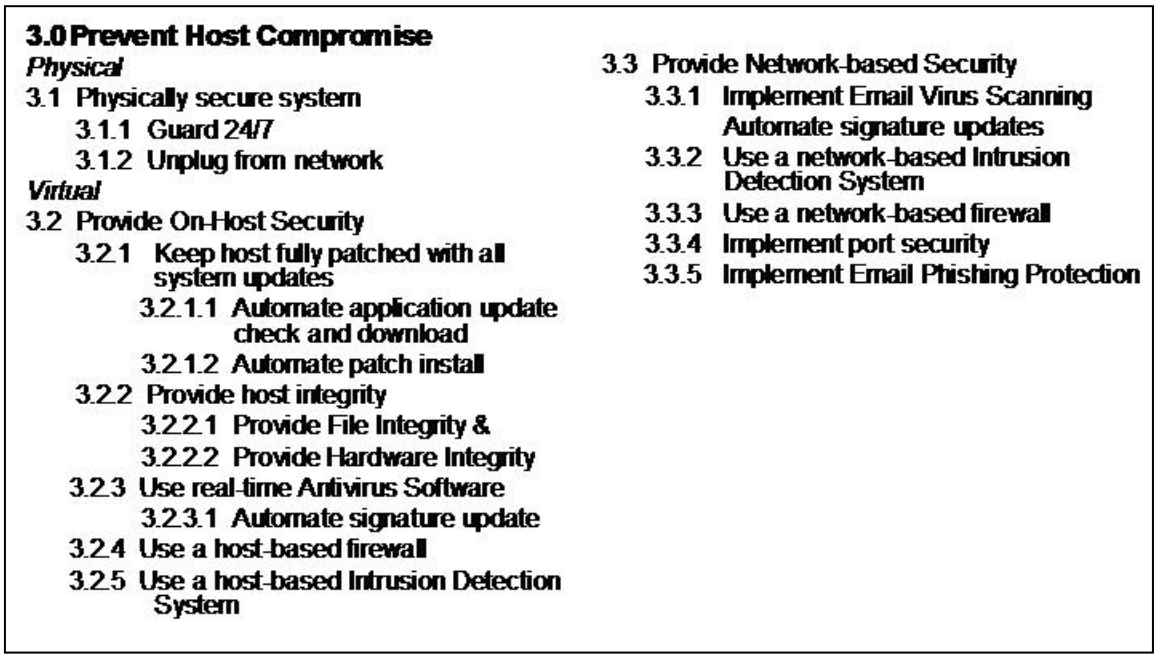

Figure 17. Defense Tree for Preventing Host Compromise

Once the defense tree is completed we can apply it to the attack tree to investigate whether it has successfully mitigated or partially mitigated the attack. As discussed in the previous chapter, this can be a serial or parallel process. In this example, the defense actions will be implemented in parallel against the attack tree to highlight mitigations each protection provides. Figure 18 shows the expected mitigation of the host attack trees using protection 3.2.2 Provide host integrity and its sub-protections.

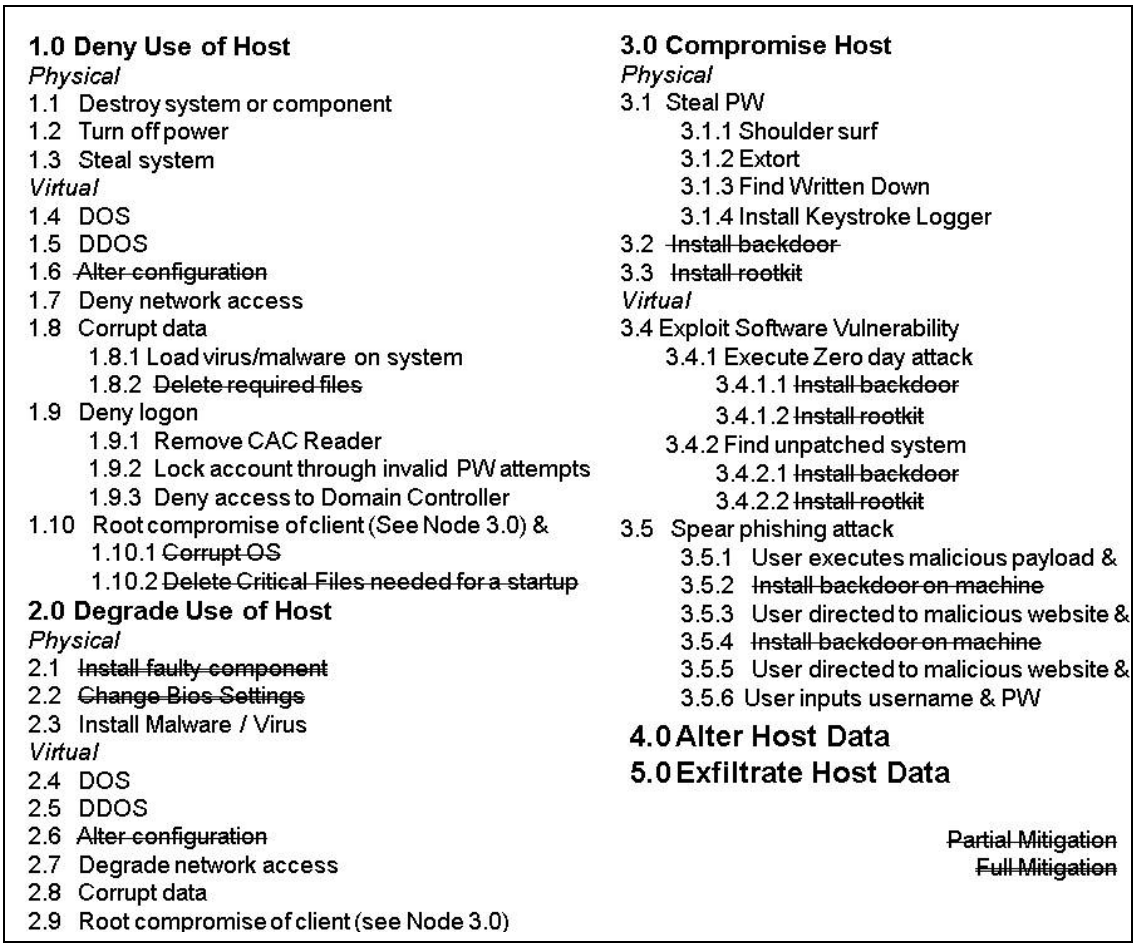

Figure 18. Resulting Attack Tree Once Host Integrity Defense Tree Applied

Figure 19 shows the expected mitigation from 3.2.1 Keep host fully patched with all system updates. The defense trees may provide full or partial mitigation to attacks. Partial mitigation would reduce the risk of a successful attack increasing the cost of the attack or by reducing either the probability of successor impact to the system. Full mitigation means that the attack is no longer possible or below a pre-determined threshold level.

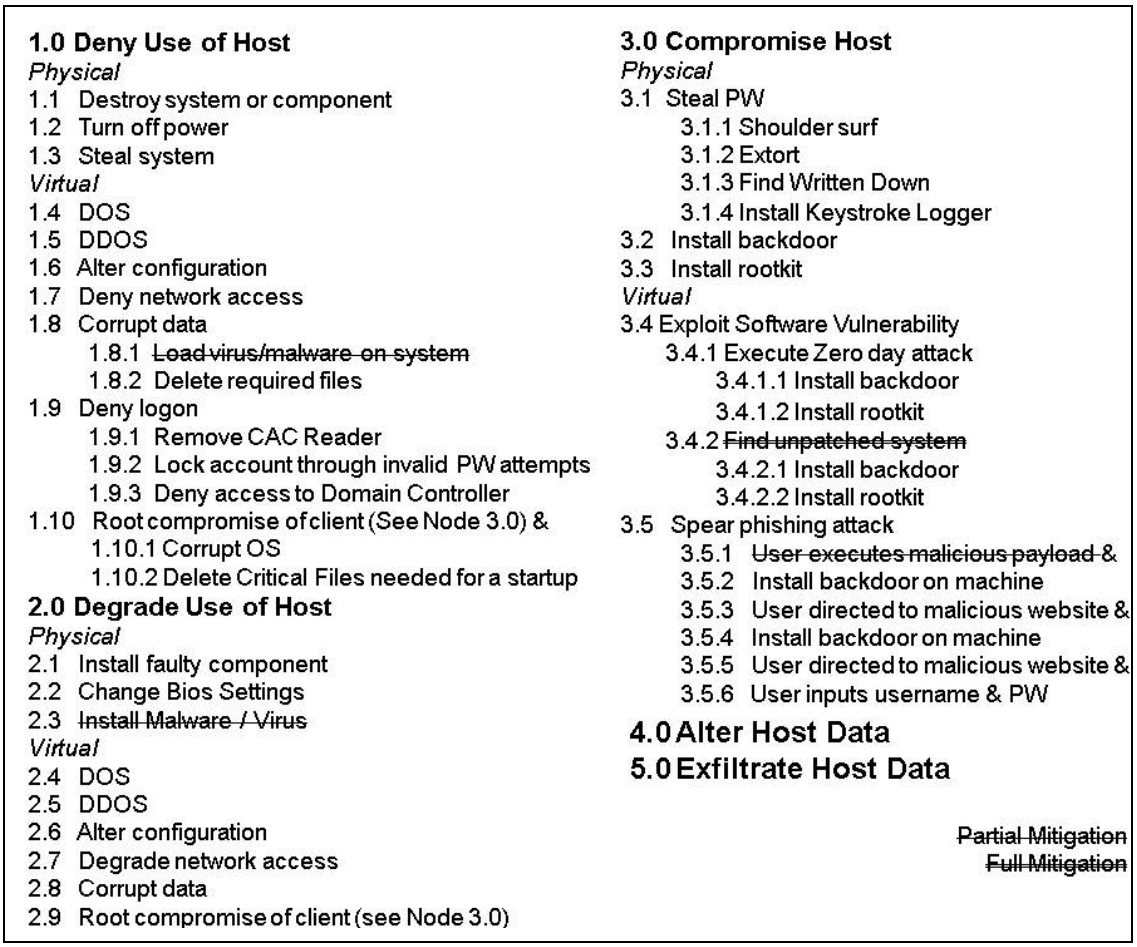

Figure 19. Resulting Attack Tree Once Keep Host Fully Patched Is Applied

Figures 20 and 21 show the notional change in the attack trees that had node values assigned. Figure 20 shows the affects of providing host integrity to the host system while figure 21 shows the affects of keeping the system fully patched. For reference both the initial node values and the new node values are included. The new node values are shown with an asterisk after the node reference number.

		Probablilty	Cost	Impact	Risk
3	**Compromise Host**				
Physical					
3.1	Steal Password				
3.1.1	Shoulder surf	5	100	6	0.30
3.1.2	Extort	20	5000	6	0.02
3.1.3	Find Written Down	5	100	6	0.30
3.1.4	Install Keystroke Logger	10	200	6	0.30
3.2	Install Backdoor	25	200	8	1.00
3.2*	Install Backdoor	15	200	7	0.53
3.3	Install Rootkit	40	200	10	2.00
3.3*	Install Rootkit	5	200	5	0.13
Virtual					
3.4	Exploit Software Vulnerability				
3.4.1	Execute Zero Day Attack	5	5000	9	0.01
3.4.1.1	Install backdoor	20	500	8	0.32
3.4.1.1*	Install backdoor	15	500	7	0.21
3.4.1.2	Install rootkit	40	500	10	0.80
3.4.1.2*	Install rootkit	5	500	5	0.05
3.4.2	Find Unpatched System	70	200	5	1.75
3.4.2.1	Install backdoor	25	500	8	0.40
3.4.2.1*	Install backdoor	15	500	7	0.21
3.4.2.2	Install rootkit	40	500	10	0.80
3.4.2.2*	Install rootkit	5	500	5	0.05
3.5	Spear phishing attack				
3.5.1	User executes malicious payload &	35	500	8	0.56
3.5.2	Install backdoor	10	500	8	0.16
3.5.2*	Install backdoor	5	500	7	0.07
3.5.3	User directed to malicious website &	50	500	8	0.80
3.5.4	Install backdoor on machine	20	500	8	0.32
3.5.4*	Install backdoor on machine	5	500	7	0.07
3.5.5	User directed to malicious website &	50	500	8	0.80
3.5.6	User inputs username & password	25	500	8	0.40

Figure 20. Resulting Attack Tree After Provide Integrity Mitigation Applied

		Probablilty	Cost	Impact	Risk
3	**Compromise Host**				
Virtual					
3.4	Exploit Software Vulnerability				
3.4.1	Execute Zero Day Attack	5	5000	9	0.01
3.4.1.1	Install backdoor	20	500	8	0.32
3.4.1.2	Install rootkit	40	500	10	0.80
3.4.2	Find Unpatched System	70	200	5	1.75
3.4.2*	Find Unpatched System	2	200	5	0.05
3.4.2.1	Install backdoor	25	500	8	0.40
3.4.2.2	Install rootkit	40	500	10	0.80
3.5	Spear phishing attack				
3.5.1	User executes malicious payload &	35	500	8	0.56
3.5.1*	User executes malicious payload &	5	2000	8	0.02
3.5.2	Install backdoor	10	500	8	0.16
3.5.3	User directed to malicious website &	50	500	8	0.80
3.5.4	Install backdoor on machine	20	500	8	0.32
3.5.5	User directed to malicious website &	50	500	8	0.80
3.5.6	User inputs username & password	25	500	8	0.40

Figure 21. Resulting Attack Tree After Patching Mitigation Applied

Both of the enumerated protection measures provided some mitigating effects on the host attack tree. These two measures should be considered as potential Cybercraft requirements. Section 4.5 will further define the elicitation process by creating use case scenarios using these two candidate requirements.

4.5 Use Case Definition and Domain Model Iterations

Once initial high-level requirements have been elicited through user involvement or attack and defense tree methodologies, it useful to create use cases to provide developers additional context of what users expect to do with the system. The requirements elicited from the attack and defense tree exercise are used to trace the process.

Two brief-format uses cases for Provide Host Integrity and Patch Host Software are shown in Figure 22 and 23 respectively. These cases are titled in typical use case format of verb noun detailing what the actor wants to use the system to do.

These brief use cases provide the highest level expression of what the system will do for the actor. They do not describe any of the preconditions, branches or fault cases which must also be taken into account in the fully-dressed use cases. The branch and fault cases are also extremely important because they specify conditions which must be handled if there is an error or decision point in the main success path. For use case one, the branch case must specify what actions should be taken if there is a difference between the host's component and the known good state or approved hardware version. Additional proposed high-level use cases are shown in Appendix B.

Use Case 1 - Provide Host Integrity (Brief/Informal)

The [Cybercraft] system will provide HOST integrity through the automated checking of HOST components against known good states. These HOST components may include files, kernel, memory, HW data structures (ACPI/Newbridge), firmware (BIOS) or peripherals (graphic cards, keyboard). If a possible integrity violation is detected, system will execute specified action.

Cybercraft PAYLOADS may be tasked to provide integrity for other nodes that may not have their own Cybercraft PLATFORM.

Figure 22. Brief Use Case Example for Providing Host Integrity

Use Case #2 –Patch Host Software (Brief/Informal)

The [Cybercraft] system will update HOST components/software through automated monitoring of HOST components and comparing them to the latest approved versions. The components may include firmware (BIOS) and drivers. If a component or software discrepancy is detected, system will execute specified action.

Cybercraft PAYLOADS may be tasked to monitor and patch other systems that may not have their own Cybercraft PLATFORM.

Figure 23. Brief Use Case Example for Patching Host Software

Once several use cases have been specified, objects emerge that will populate our domain model. Use cases one and two describe a Cybercraft system composed of platforms and payloads, a host composed of files, kernels, firmware, and peripherals, "approved versions" and "known good states." The level 0 domain model (Figure 24) is the top-level domain model for the system. Level 0 objects include host, Cybercraft platform, Cybercraft payload, network, operating system, and application. These are derived from the use cases and an understanding of the Cybercraft operating environment. This initial top-level model is useful for conveying additional context for high-level use cases. It is likely incomplete or inaccurate and will continue to evolve throughout the process to reflect additional detail as new use cases are enumerated.

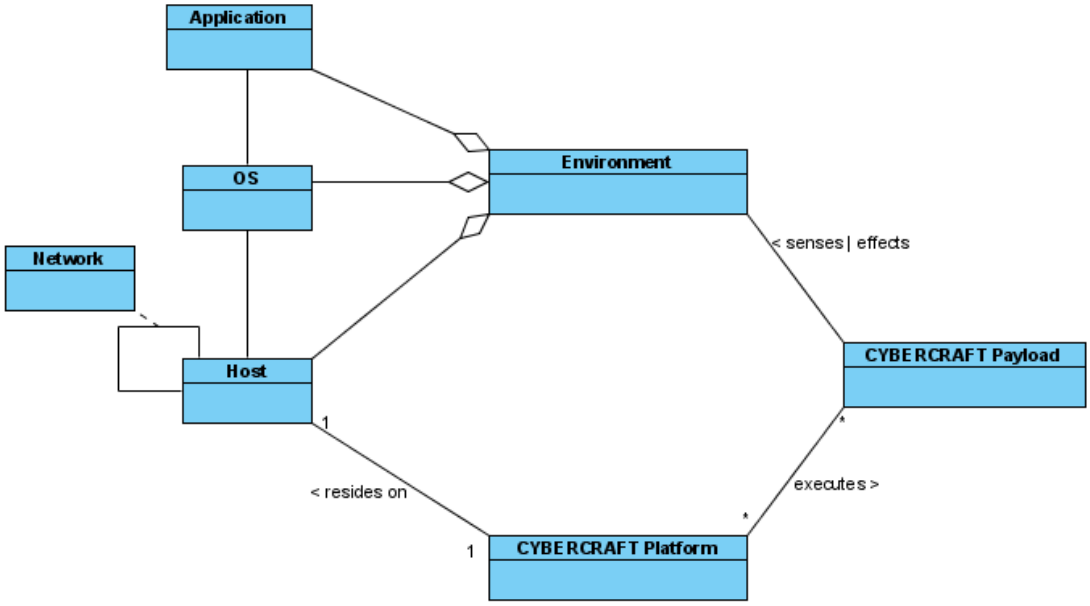

Figure 24. Level 0 Domain Model

Although not a true domain model, it is also useful to create a high-level, user view of the system. Operator, commander, technician, and developer are several of the possible Cybercraft user categories defined through an understanding of the operating environment, architecture and use cases. Each of these types of users will have different requirements for what Cybercraft will do and how they need to interface with it. As shown in Figure 25, this is not a user interface specification, but a systems view.

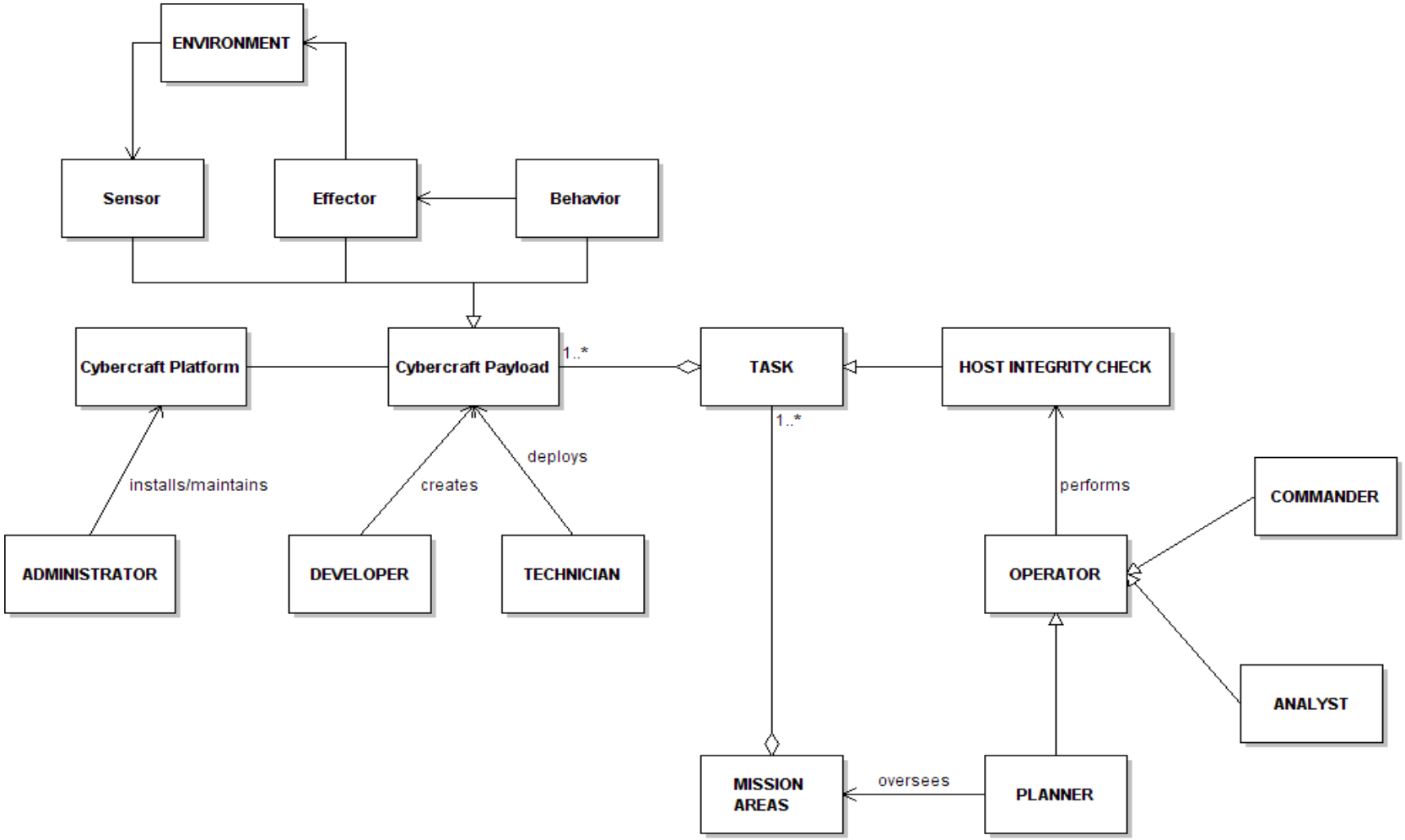

Figure 25. User View of the System

Once these initial views have been created, the use cases and domain models design process should begin to build additional depth. There are several options when deciding which use cases to expand to provide additional system fidelity. The options include starting with the use cases which may be pre-conditions for other use cases or those which have the highest level of user interest. Another option, which will be used in this example, is to pick the complex use cases which will provide fidelity to the largest portion of the system. This approach facilitates greater detail system more quickly than choosing only pre-condition use case because more complex use cases typically exercise more system components.

A formal uses case provides the typical success scenario path in full detail. It also provides the decision branches and failures that may occur. Figure 26 contains parts of the Provide Host Integrity use case (full version in Appendix C) using a formal use case template. Preconditions are those things which must be true for this use case to be valid. Preconditions typically form the basis for other use cases. In the Provide Host Integrity example, both the Cybercraft platform and all required must be loaded on the host.

Another important aspect of the formal use case is the description of the trigger that begins this use case. In this case, an operator will load the task on a Cybercraft. This task includes the system components to monitor, the known good representations for the component, and actions to take depending on success or failure of comparison between known good representation and current representation.

USE CASE #	1. Provide Host Integrity	
Goal in Context	System will provide through the automated checking of system components (files, kernel, memory, HW data structures, firmware, or peripherals	
Preconditions	Cybercraft platform loaded on host All required cybercraft payloads required for have been loaded onto the Cybercraft platform	
Success End Condition	Cybercraft system continues to monitor system components' integrity	
Failed End Condition	Cybercraft system logs system state and alerts operator	
Primary, Secondary Actors	Operator	
Trigger	Operator executes Provide Integrity Task on Cybercraft system Task composed of list of system components, known good states representation for components, and actions to take on success and failure	
DESCRIPTION	**Step**	**Action**
	1	Sensor payload reads current component state
	2	Sensor payload creates current component state representation
	3	Sensor payload stores current component state representation in State
	4	Behavior payload reads component representation from State
	5	Behavior payload compares current state representation to stored "known good" representation
	6	Component state representation matches "known good" representation
EXTENSIONS	**Step**	**Branching Action**
System Fails	1a	Sensor payload cannot read component state
System Fails	2a	Sensor payload cannot create state representation
System Fails	3a	Sensor payload cannot store representation in State
System Fails	4a	Behavior payload cannot read representation in State
System Fails	5a	Behavior payload cannot compare representations
		For all of the branches above, failure actions are executed
	6a	Component state representation does not match "known good" representation
	6b	Effector takes specified actions

Figure 26. Formal Use Case for Providing Host Integrity

The next section of the formal use case contains the detailed, step by step description of the typical success path. Additional the branches and failure cases are also shown. The typical success path in our system is that there has been no compromise of host integrity. Our only non-failure case details what should happen if host integrity is compromised.

Once formal use cases have been developed that provided expanded understanding of the domain, it is necessary to either update an existing domain model or create another domain model with the additional detail. Figure 27 shows the level one domain model expanded with objects described in the use case. The effector, behavior, and sensor payloads have been added as well as the (system) state. The components (files, kernel, memory, etc) Cybercraft will monitor are also shown.

The process of use case and domain model iteration should continue until the system has been defined to such a level that a system prototype can be created. System design abstraction can also provide additional detail. Figure 28 shows the Cybercraft system in additional detail with the controller, sequencer, and coordinator necessary to handle all of the tasks defined in the formal version of the Provide Host Integrity use case. The level two domain model has been decomposed enough that prototype development can begin to be contemplated.

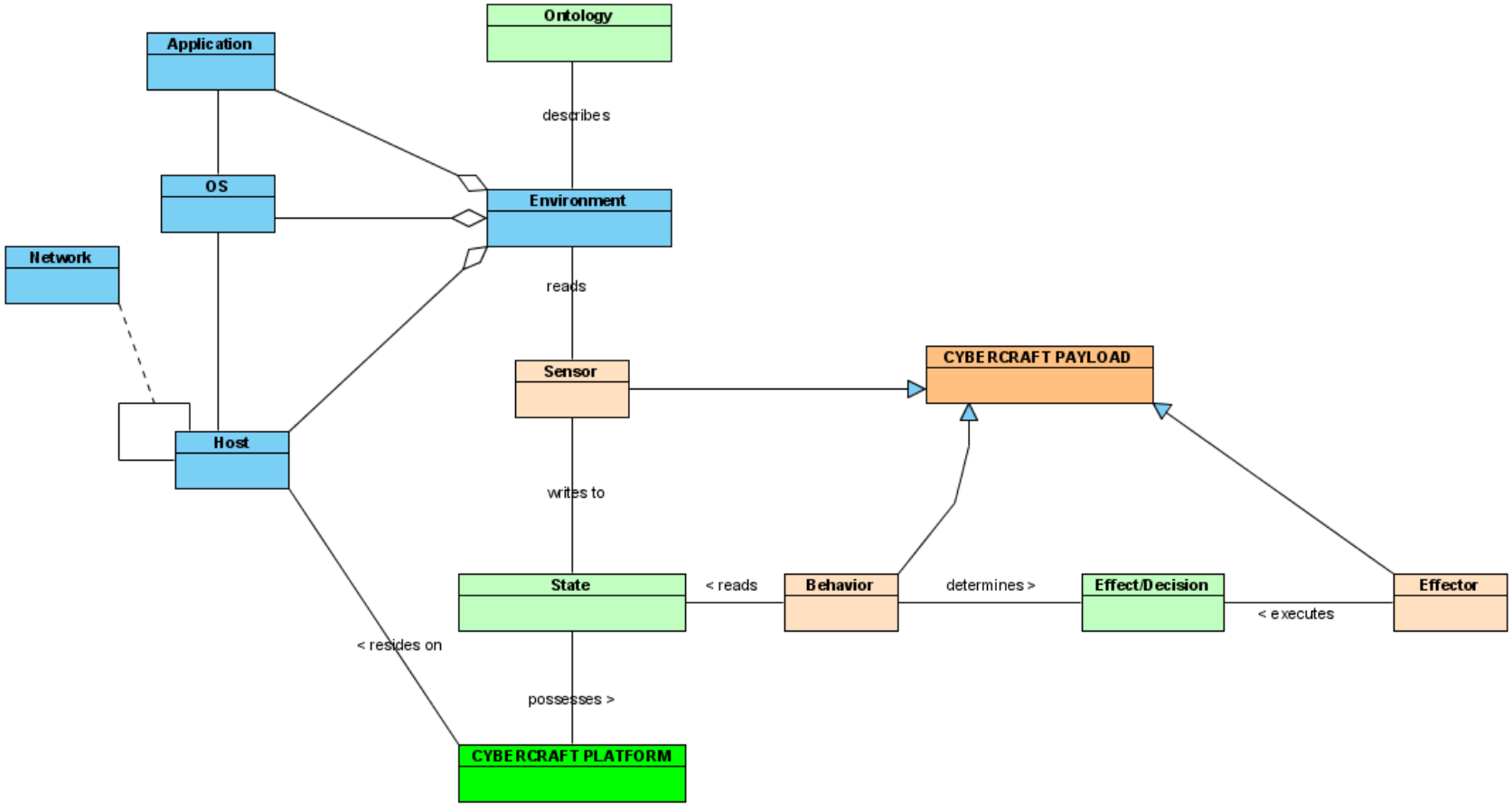

Figure 27. Level 1 Domain Model

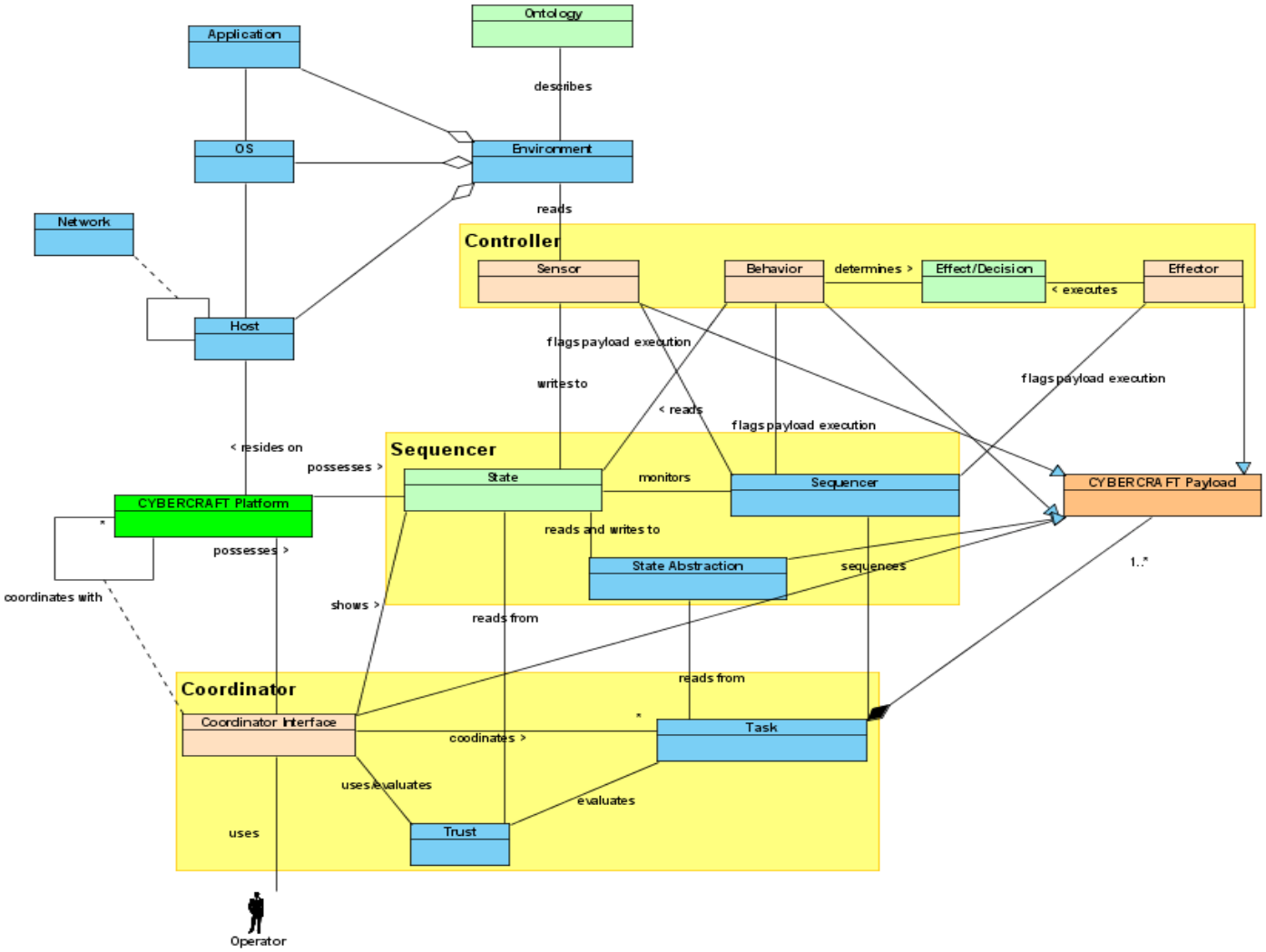

Figure 28. Level 2 Domain Model

4.6 Summary

This chapter provided the results and analysis for the application of the proposed Cybercraft requirements development methodology. The need for an overarching vision and scope document was identified to ensure all users and developers clearly understand the development goals and what the system will provide to its users. Several tools and techniques were also used to elicit requirements including applying attack and defense trees based on the current and emerging threat environment. Finally, use cases were developed in concert with a domain model which if iterated could provide the fidelity required to begin design tradeoffs and prototype development.

V. CONCLUSION AND FUTURE WORK

Defining the requirements for the next-generation network defense tool is critically important to the overall success of the Cybercraft program. Cybercraft is an ambitious project in a complex and highly dynamic environment being designed through a distributed working group structure across different organizations and geographic areas. These factors only increase the complexity.

The methodology presented in this document describes how to develop system requirements for Cybercraft. This was accomplished by examining the threat environment using attack trees and using that information to create defense trees that might mitigate those attacks. The methodology also incorporated the software design technique of use cases to describe the system in terms of what it might do for an actor using the system and domain models to present a visual understanding of the system that once fully defined could be used to generate code. The continuing work will need to build upon this process to fully describe Cybercraft requirements and transfer the domain models into prototypes which can be built and tested.

5.1 Future Work

The Cybercraft project is still in the beginning stages of developing the specifications needed to design and implement the Cybercraft network defense system. This section discusses recommendations for future work that will support the goal of creating the Air Forces next-generation network defense system.

The Cybercraft project is an ambitious project born out of a realization that current network defense systems are inadequate to defeat current and emerging threats. Its goal of providing a hardware system which scales up to one million nodes will have a significant cost and logistics requirement when implemented. An initial feasibility study should be conducted to determine order of magnitude costs and logistical requirements to implement the architectural vision for Cybercraft.

Current threat models and attack trees for enterprise-level networks similar in size to the Air Force network do not exist. While the Department of Homeland Security attack patterns library is available, it is in its infancy. It is also a time-consuming manpower-intensive process. Developing a comprehensive enterprise attack tree would provide numerous programs and projects insight into the threat environment systems are being designed to defeat or into which they will be forced to operate.

Appendix A. Attack Trees

Host Attacks

1.0 Deny Use of Host
- Physical
  - 1.1 Destroy system or component
  - 1.2 Turn off power
  - 1.3 Steal system
- Virtual
  - 1.4 DOS
  - 1.5 DDOS
  - 1.6 Alter configuration
  - 1.7 Deny network access
  - 1.8 Corrupt data
    - 1.8.1 Load virus/malware onto system
    - 1.8.2 Delete required files
  - 1.9 Deny logon
    - 1.9.1 Remove CAC Reader
    - 1.9.2 Lock account through invalid PW attempts
    - 1.9.3 Deny access to Domain Controller
  - 1.10 Root compromise of client (See Node 3.0)
    - 1.10.1 Corrupt OS
    - 1.10.2 Delete critical files needed for startup

2.0 Degrade Use of Host
- Physical
  - 2.1 Install faulty component
  - 2.2 Change Bios Settings
  - 2.3 Install Malware / Virus
- Virtual
  - 2.4 DOS
  - 2.5 DDOS
  - 2.6 Alter configuration
  - 2.7 Degrade network access
  - 2.8 Corrupt data
  - 2.9 Root compromise of client (see Node 3.0)

3.0 Compromise Host
- Physical
  - 3.1 Steal PW
    - 3.1.1 Shoulder surf
    - 3.1.2 Extort
    - 3.1.3 Find Written Down

3.1.4 Install Keystroke Logger
3.2 Install backdoor
3.3 Install rootkit
Virtual
3.4 Exploit Software Vulnerability
3.4.1 Execute Zero day attack
3.4.1.1 Install backdoor
3.4.1.2 Install rootkit
3.4.2 Find unpatched system
3.4.2.1 Install backdoor
3.4.2.2 Install rootkit
3.5 Spear phishing attack
3.5.1 User executes malicious payload &
3.5.2 Install backdoor
3.5.3 User directed to malicious website &
3.5.4 Install backdoor
3.5.5 User directed to malicious website &
3.5.6 User inputs username and password

4.0 Alter Host Data

5.0 Exfiltrate Host Data

Network Infrastructure Attacks

6.0 Deny Use of Network
Physical
6.1 Destroy node or component
6.2 Inhibit communication media
6.2.1 Cut cable
6.2.2 Jam frequency
6.3 Turn off power
6.3.1 Node
6.3.2 Repeater
6.4 Steal node
Virtual
6.5 DOS
6.6 DDOS
6.7 Alter node configuration
6.8 Compromise network node (see 8.0)

7.0 Degrade Use of Network
- Physical
- 7.1 Install faulty component
- 7.2 Damage communication media
  - 7.2.1 Damage cables
  - 7.2.2 Jam frequency
- 7.3 Alter node configuration
- Virtual
- 7.4 DOS
- 7.5 DDOS
- 7.6 Alter node configuration
- 7.7 Inject data errors
- 7.8 Root compromise of network node (see Node 8.0)

8.0 Compromise network node
- Physical
- 8.1 Steal PW
  - 8.1.1 Shoulder surf
  - 8.1.2 Extort
  - 8.1.3 Find Written Down
  - 8.1.4 Install Keystroke Logger
- 8.2 Load alternate configuration
- Virtual
- 8.3 Exploit Software Vulnerability
  - 8.3.1 Zero-day attack
  - 8.3.2 Unpatched System
  - 8.3.3 Upgrade privileges to admin/root
  - 8.3.4 Install alternate configuration

9.0 Alter Network Data
- 9.1 Man In the Middle Attack

10.0 Exfiltrate Network Data
- 10.1 Compromise node &
- 10.2 Install alternate configuration redirecting traffic

Additional Categories to Complete

Attacks against a System (ex. Global Command and Control System (GCCS), Military Personnel Data System (MilPDS)

Network (Local, base, AF Enterprise)

Appendix B. Proposed High-Level Use Cases

Use Case 1 - Provide Host Integrity (Brief/Informal)

The [Cybercraft] system will provide HOST integrity through the automated checking of HOST components against known good states. These HOST components may include files, kernel, memory, HW data structures (ACPI/Newbridge), firmware (BIOS) or peripherals (graphic cards, keyboard). If a possible integrity violation is detected, system will execute specified action.

Cybercraft PAYLOADS may be tasked to provide integrity for other nodes that may not have their own Cybercraft PLATFORM.

Use Case #2 –Patch Host Software (Brief/Informal)

The [Cybercraft] system will update HOST components/software through automated monitoring of HOST components and comparing them to the latest approved versions. The components may include firmware (BIOS) and drivers. If a component or software discrepancy is detected, system will execute specified action.

Cybercraft PAYLOADS may be tasked to monitor and patch other systems that may not have their own Cybercraft PLATFORM.

Use Case #3 –Install Cybercraft Platform (Brief/Informal)

An administrator will install a Cybercraft PLATFORM on a HOST or NETWORK NODE. PLATFORM will report successful installation to CONTROL SYSTEM.

Use Case #4 –Deploy Cybercraft Payload (Brief/Informal)

The Cybercraft CONTROL SYSTEM will deploy a Cybercraft PAYLOAD to a Cybercraft PLATFORM. PLATFORM will report successful installation to CONTROL SYSTEM.

Use Case #5 –Execute Cybercraft Task (Brief/Informal)

An authorized OPERATOR will select a TASK for one or more Cybercraft PLATFORMS using the CONTROL SYSTEM. CONTROL SYSTEM will load required PAYLOADS onto Cybercraft PLATFORM(S). CONTROL SYSTEM will provide Cybercraft PLATFORM with TASK instructions.

Use Case #6 –Detect Host Intrusion from Known Signatures (Brief/Informal)

The [Cybercraft] system will monitor HOST network traffic for possible system intrusion. System will compare host network traffic with known attack signatures. If a possible intrusion is detected, system will execute specified action.

Cybercraft PAYLOADS may be tasked to monitor and detect intrusions on other systems that may not have their own Cybercraft PLATFORM.

Use Case #7 –Detect Host Intrusion from Anomalous Behavior Analysis (Brief/Informal)

The [Cybercraft] system will monitor HOST components for possible system intrusion. System will compare host network traffic with system usage rules or learned patterns of behavior. If a possible intrusion is detected, system will execute specified action.

Cybercraft PAYLOADS may be tasked to monitor and detect intrusions on other systems that may not have their own Cybercraft PLATFORM.

Use Case #8 –Maintain/Perform Host Configuration Management (Brief/Informal)

The [Cybercraft] system will maintain a complete inventory of HOST components. These HOST components may include software applications, firmware (BIOS) or installed hardware peripherals (graphic cards, keyboard). System may provide information to a centralized CONTROL SYSTEM. If a change in system configuration is detected, system will execute specified action.

Cybercraft PAYLOADS may be tasked to monitor and perform configuration on other systems that may not have their own Cybercraft PLATFORM.

Use Case #9 –Reconfigure Host Network Address (Brief/Informal)

The [Cybercraft] system will dynamically change the network address [IP, MAC?] of its HOST.

Cybercraft PAYLOADS may be tasked to change the network address on other systems that may not have their own Cybercraft PLATFORM

Appendix C. Proposed Formal Use Case

USE CASE #	1. Provide Host Integrity	
Goal in Context	System will provide through the automated checking of system components (files, kernel, memory, HW data structures, firmware, or peripherals)	
Scope & Level	Primary Task	
Preconditions	Cybercraft platform loaded on host All required Cybercraft payloads required for have been loaded onto the Cybercraft platform	
Success End Condition	Cybercraft system continues to monitor system components' integrity	
Failed End Condition	Cybercraft system logs system state and alerts control system	
Primary, Secondary Actors	Operator	
Trigger	Cybercraft system is given the task to monitor system components Task composed of list of system components, known good states representation for components, and actions to take on success and failure	
DESCRIPTION	**Step**	**Action**
	1	Sensor payload reads current component state
	2	Sensor payload creates current component state representation
	3	Sensor payload stores current component state representation in State
	4	Behavior payload reads component representation from State
	5	Behavior payload compares current state representation to stored "known good" representation
	6	Component state representation matches "known good" representation
EXTENSIONS	**Step**	**Branching Action**
System Fails	1a	Sensor payload cannot read component state
System Fails	2a	Sensor payload cannot create state representation
System Fails	3a	Sensor payload cannot store representation in State
System Fails	4a	Behavior payload cannot read representation in State
System Fails	5a	Behavior payload cannot compare representations
		For all of the branches above, failure actions are executed
	6a	Component state representation does not match "known good" representation
	6b	Effector takes specified actions

		Branching Action
SUB-VARIATIONS		
RELATED INFORMATION		
Priority:	Top	
Performance	< 2 minutes	
Frequency	Every 60 minutes	
Channels to actors	Interactive, database	
OPEN ISSUES		
Due Date		
...any other management information...		
Superordinates		
Subordinates	Install Cybercraft platform Deploy Cybercraft payload Execute Cybercraft task	

Bibliography

Adolph, S., & Bramble, P. (2003). *Patterns for Effective Use Cases*. Boston: Addison-Wesley.

Bibighaus, D. (Sep 2006). *Cybercraft Whitepaper.* Air Force Research Lab. Rome New York.

Bittner, K., & Spence, I. (2003). *Use Case Modeling*. Boston, MA: Addison Wesley.

Boehm, B. W. (1988). A Spiral Model of Software Development and Enhancement. *Computer, 21*(5), 61-72.

Brooks, F. P., Jr. (1987). No Silver Bullet Essence and Accidents of Software Engineering. *Computer, 20*(4), 10-19.

Cartwright, James (2007). Statement On United States Strategic Command Before The House Armed Services Committee: Armed Services Committee, United State House of Representatives, 21 March 2007.

Cockburn, A. (2001). *Writing Effective Use Cases*. Boston: Addison-Wesley.

Daley, K. (2002). A Structural Framework For Modeling Multi-Stage Network Attacks. *Parallel Processing Workshops, 2002. Proceedings. International Conference on,* 5-10.

Department of Homeland Security National Cyber Security Division. (2008a). *Build security in.* Retrieved May 8, 2008, from https://buildsecurityin.us-cert.gov/daisy/bsi/home.html.

Department of Homeland Security National Cyber Security Division. (2008b). *Common attack pattern enumeration and classification.* Retrieved May 8, 2008, from http://capec.mitre.org/

Edge, K. S. (2006). Using Attack And Protection Trees To Analyze Threats And Defenses To Homeland Security. *Military Communications Conference, 2006. MILCOM 2006,* 1-7.

Glumich, S. (2008). *Cybercraft Overview:* Spring 2008 Cybercraft Workshop. Wright Patterson Air Force Base, OH.

Goldman, H. G., & Woodward, John P. L. (January 2008). *Defending Against Advanced Cyber Threats.* The MITRE Corporation.

Hunt, S. (2008). *Developing A Reference Framework For Cybercraft Trust Evaluation.* MS Thesis, AFIT/GCS/ENG/08-11. School of Electrical and Computer Engineering, Air Force Institute of Technology (AU), Wright Patterson Air Force Base, OH.

Institute for Electrical and Electronics Engineers (IEEE). (1991). *IEEE Standard Computer Dictionary. A Compilation of IEEE Standard Computer Glossaries.*

Jacobson, I., Booch, G., & Rumbaugh, J. (1998). *The Unified Software Development Process*. Reading, Mass.; Harlow: Addison-Wesley.

Larman, C. (2003). Iterative And Incremental Developments. A Brief History. *Computer, 36*(6), 47-56

Larman, C. (2005). *Applying UML and patterns : An introduction to object-oriented analysis and design and iterative development.* Upper Saddle River, N.J.: Pearson Education International.

Mauw, S., & Oostdijk, M. (2006). Foundations of Attack Trees. *Lecture Notes in Computer Science.*, (3935), 186-198.

McCracken, D. D., & Jackson, M. A. (1982). Life Cycle Concept Considered Harmful. *SIGSOFT Software.Engineering Notes, 7*(2), 29-32.

McDonald, J. T. (2007). *Requirements Working Group Session.* Fall 2007 Cybercraft Workshop. Air Force Research Labs Rome, New York.

McDonald, J. T. (2008). *Cybercraft Requirements Working Group Presentation.* Spring 2008 Cybercraft Workshop. Wright Patterson Air Force Base, OH.

Myagmar, S., Lee, A., & Yurcik, W. (2005). Threat Modeling As A Basis For Security Requirements. *Symposium on Requirements Engineering for Information Security (SREIS).*

Phister, P. W., Fayette, D., & Krzysiak, E. (2005). The "Cybercraft" Concept Linking Ncw Principles With The Cyber Domain In An Urban Operational Environment. *The Future of Command and Control 10th ICCRTS, June 13-16, 2005, McLean, VA, 11*, 234-242.

Rosenberg, D., & Scott, K.,. (1999). *Use Case Driven Object Modeling With UML : A Practical Approach*. Reading, MA: Addison-Wesley.

Royce, W. W. (1987). Managing The Development Of Large Software Systems: Concepts And Techniques. *ICSE '87: Proceedings of the 9th International Conference on Software Engineering,* Monterey, California, United States. 328-338.

Schneier, B. (1999). Attack Trees. *Dr.Dobb's Journal, 24*(12), 21-29.

Schneier, B. (2008). *Inside The Twisted Mind Of The Security Professional.* Retrieved May 8, 2008, from http://www.wired.com/politics/security/commentary/securitymatters/2008/03/securitymatters_0320.

Schneier, B.,. (2000). *Secrets And Lies: Digital Security In A Networked World*. New York; Chichester: Wiley.

Smith, L. W. (2000). Project Clarity Through Stakeholder Analysis. *Crosstalk: The Journal of Defense Software Engineering,* (12)

Sommerville, I., & Sawyer, P. (2000). Requirements Engineering: A Good Practice Guide. *European Journal Of Information Systems : An Official Journal Of The Operational Research Society., 9,* 124.

Swiderski, F., & Snyder, W. (2004). *Threat Modeling*. Redmond, WA, USA: Microsoft Press.

Torr, P. (2005). Demystifying The Threat Modeling Process. *Security & Privacy, IEEE, 3*(5), 66-70.

United States Air Force Scientific Advisory Board. (2007). *Report On The Implications Of Cyber Warfare, Volume 2: Final Report* No. SAB-TR-07-02). Washington, DC:

Wiegers, K. E. (2003). *Software Requirements : Practical Techniques For Gathering And Managing Requirements Throughout The Product Development Cycle* (2nd ed.). Redmond, Wash: Microsoft Press.

Wiegers, K. E. (2006). *More About Software Requirements : Thorny Issues And Practical Advice*. Redmond, WA: Microsoft Press.

Vita

Major Michael Hunsberger grew up in Perkasie, Pennsylvania graduating from Pennridge High School. He attended Rochester Institute of Technology in Rochester, New York where he graduated with a Bachelor of Science degree in Computer Engineering. He was commissioned through the Air Force ROTC program at Detachment 538. Major Hunsberger is a career communications and information officer.

His first assignment was at Tinker Air Force Base where he was a communications engineer in the 38th Engineering and Installation Group. Major Hunsberger has held assignments at Osan Air Base, Republic of Korea, the Air Force Communications Agency at Scott Air Force Base, and Elmendorf Air Force Base where he was a the 3rd Mission Support Group executive officer and 3rd Communications Squadron information systems flight commander and director of operations.

Major Hunsberger attended the Naval Postgraduate School in Monterey California where he was a distinguished graduate and earned a Master of Science in Systems Technology and a Master of Science in Computer Science.

Major Hunsberger has also had three deployments to Prince Sultan Air Base in 1997, Joint Task Force Operation Northern Watch at Incirlik Air Base Turkey in 2002, and Joint Task Force 536 Utapao, Thailand in 2006 supporting tsunami relief efforts.

In May 2007, Major Hunsberger entered the Graduate School of Engineering and Management, Air Force Institute of Technology. Upon graduation, he will be the commander, Detachment 2, 561st Network Operations Squadron at Randolph Air Force Base.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. REPORT DATE *(DD-MM-YYYY)*	**2. REPORT TYPE**	**3. DATES COVERED** *(From – To)*
06-19-2008	**Master's Thesis**	May 2007 –Jun 2008

4. TITLE AND SUBTITLE

A Methodology for Cybercraft Requirement Definition and Initial System Design

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Hunsberger, Michael G., Major, USAF

5d. PROJECT NUMBER
08-198

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(S)
Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT/ICW/ENG/08-03

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Ms. Sonja Glumich, Sonja.Glumich@rl.af.mil, 315-330-4370
Air Force Research Laboratory
Information Directorate
525 Brooks Rd,
Rome, New York 13441

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
The United States Air Force and Department of Defense networks and information system are under attack from a variety of actors. Current network defense systems are reactive in nature and unable to prevent determined adversaries from successfully infiltrating these information systems. The realization of these facts led the Air Force Research Lab begin work on a next-generation network defense system called Cybercraft. The Cybercraft vision is a trusted, autonomous system which will perform network defense tasks.

In this paper, software engineering and threat analysis are used to create a set of initial requirements and system models for Cybercraft. This paper presents a methodology based on traditional software requirements elicitation processes and attack and defense trees to generate system requirements. Once requirements have been defined, they are used to create system use cases and a system domain model. This iterative process can be used to define the system in enough detail that software or system prototypes can be developed. The contribution of this paper is a set of initial requirements, use cases, and domain models which could be used in Cybercraft development. Ultimately, it is a generic methodology which could be used to determine requirements for any security system and how to apply those requirements to begin high-level system design.

15. SUBJECT TERMS
Requirements, network security, model, network architecture
Cybercraft, attack tree, defense tree, use case, domain model

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
REPORT	**ABSTRACT**	**c. THIS PAGE**			Lt Col Todd McDonald (ENG)
U	**U**	**U**	**UU**	73	**19b. TELEPHONE NUMBER** *(Include area code)* (937) 255-6565, ext 4314; e-mail: Jeffrey.mcdonald@afit.edu

Standard Form 298 (Rev: 8-98)
Prescribed by ANSI Std. Z39-18