Threat Modeling as a Basis for Security Requirements

Suvda Myagmar Adam J. Lee William Yurcik

National Center for Supercomputing Applications (NCSA)
University of Illinois at Urbana-Champaign
{myagmar, adamlee, byurcik}@ncsa.uiuc.edu

Abstract

We routinely hear vendors claim that their systems are "secure." However, without knowing what assumptions are made by the vendor, it is hard to justify such a claim. Prior to claiming the security of a system, it is important to identify the threats to the system in question. Enumerating the threats to a system helps system architects develop realistic and meaningful security requirements.

In this paper, we investigate how threat modeling can be used as foundations for the specification of security requirements. Although numerous works have been published on threat modeling, there is a lack of integrated, systematic approach toward threat modeling for complex systems. We examine the differences between modeling software products and complex systems, and outline our approach for identifying threats of networked systems. We also present three case studies of threat modeling: Software-Defined Radio, a network traffic monitoring tool (VisFlowConnect), and a cluster security monitoring tool (NVisionCC).

1. Introduction

It is widely accepted that designing secure computer systems is a difficult problem. Attackers routinely break into systems and in response, software vendors started providing security as a necessary feature for their products and network systems. As a result of years of research, many powerful techniques have been developed to solve a wide array of security problems. While claiming that a system leverages a "public-key infrastructure" or uses "128-bit keys" sounds impressive, these statements mean very little when taken out of context. The important question to ask is "Are the security features of the system necessary, and do they meet the system's security needs?"

When choosing security measures, the security system designer must consider the design of the entire system, and not incorporate security technologies at random. In [8], Bruce Schneier states that "Security is a chain; it's only as secure as the weakest link. Security is a process, not a product." Designing system security is best done by utilizing a systematic engineering approach. *Systems security engineering* is concerned with identifying security risks, requirements and recovery strategies [5]. It involves well defined processes through which designers develop security mechanisms. Ideally, security engineering should be incorporated into the system design process as early as possible, from the initial architecture specification, if possible. The earlier security concerns are addressed, less time consuming and costly it is to fix future security problems. Despite this well-known fact, it is often the case that software engineers find themselves needing to retrofit security into an existing system. In either case, the security engineering process may be applied in similar manner.

One view of security engineering process is given in Figure 1. Threat modeling involves understanding the complexity of the system and identifying all possible threats to the system, regardless of whether or not they can be exploited. During the formation of security requirements, these threats are analyzed based on their criticality and likelihood, and a decision is made whether to mitigate the threat or accept the risk associated with it. Once system designers determine which security mechanisms must be available to the system, the development of these mechanisms follows the general software engineering cycle of design, implementation, testing, and maintenance. Each stage of security engineering feeds back to the preceding stage, and through that stage to all earlier stages. Feedback allows designers to catch mistakes made in the early stages without letting their effects cascade. Threat modeling and security requirements provide the foundations upon which the rest of the security system is built.

Identifying threats helps develop realistic and meaningful security requirements. This is particularly important, for if the security requirements are faulty, the definition of security for that system is faulty, and thus the system cannot be secure. Proper identification of threats and appropriate selection of countermeasures reduces the ability of attackers to misuse the system. In that respect, threat mod-

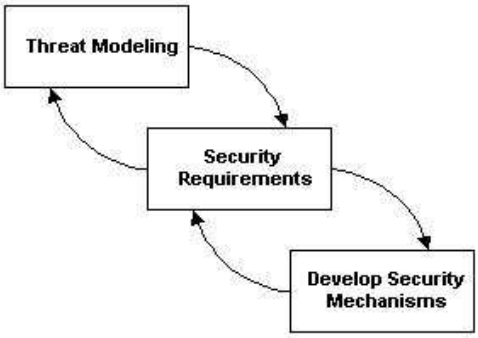

Figure 1. System security engineering

eling looks at the system from an adversary's perspective to help designers anticipate attack goals and determine answers to questions about *what* the system is designed to protect, and from *whom*. Any type of system can benefit from threat modeling. Some systems are fairly simple and others are more complex, some of them are already deployed and others exist only on paper. No matter what the system is or what stage of the development process it is in, the benefits from well thought out threat model can prove extremely useful. We elaborate on the different approaches toward threat modeling with respect to these varying types of systems later in the paper.

There are several existing works on threat modeling and many more on requirements engineering. In this paper, we build on these bodies of work by investigating how threat modeling can be used as a basis for the specification of security requirements. We advocate the use of threat modeling as an important step toward addressing the completeness of the system requirements in general, and security requirements in particular. Threat modeling can be used to justify security countermeasures and validate the assumptions made by system architects. We discuss the interplay between threat modeling and requirements definition and examine the differences between modeling software products and complex systems. We also present three case studies of threat modeling: Software-Defined Radio, a network traffic monitoring tool, and a cluster security monitoring tool.

The remainder of this paper is organized as follows. In Section 2 we discuss the threat modeling process and how it differs for software products and complex systems. We then explain how security requirements are specified based on a threat model in Section 3. We present our case studies in Section 4 and conclude with a summary in Section 5.

2. Threat Modeling Process

A good threat model allows security designers to accurately estimate the attacker's capabilities. It might be tempting to skip threat modeling and simply extract the system's security requirements from "industry's best practices" or standards such as Common Criteria [2]. However, these standards merely provide general security guidance and cannot address all of the nuances of a particular system. The common standards almost always need some customization for the target system and additional requirements need to be defined. Jeanette Wing *et al.* [9, 10] stress the use of attack graphs in determining what security measures to deploy in a system. However, attack trees model a chosen set of attacks via a finite state machine and are feasible only in small scenarios. Moreover, a list of potential threats still needs to be compiled prior to generating attack trees. Thus, threat modeling is required for:

- Complex software systems that integrate multiple infrastructures and technologies
- Customized application solutions
- All other cases where it is unacceptable to implement pre-compiled "to-do" lists provided by a software vendor or standards committee

A threat model cannot be created by simply brainstorming an adversary's possible intentions. This approach is not systematic and is likely to leave large portions of the attackspace uninvestigated. An attacker only has to find one security flaw to compromise the whole system [7]. Thus, it is important to be systematic during the threat modeling process to ensure that as many possible threats and vulnerabilities are discovered by the developers, not the attackers. Threat analysis should be used in the earliest possible stages of system design. Although the effort required to threat model an existing system is the same as for threat modeling a system during its early design stages, it is harder and costlier to mitigate the threats detected in an existing system due to architectural constraints.

Our threat modeling process consists of the following three high-level steps: *characterizing the system*, *identifying assets and access points*, and *identifying threats*. Although this process may appear similar to the one presented by Swiderski and Snyder [11], the sequence and description of steps is different and the execution of steps is extended to suit complex, networked systems. Their threat modeling process targets software applications.

Characterizing the system involves understanding the system components and their interconnections, and creating a system model emphasizing its main characteristics. Then assets and access points of the system are identified. Identifying threats creates a threat profile of a system, describing all the potential attacks that need to be mitigated against or accepted as low risk.

Although these three steps of threat modeling process are common to all type of systems, the actual execution of these

steps differs depending on the type of the system. An application and a networked system are represented by completely different system models in the characterization step. An application is modeled by a Data Flow Diagram, and a networked system by a Network Model. The consecutive threat modeling steps apply to these varying system models. Next, we elaborate on each of these threat modeling steps.

2.1. Characterizing the System

At the start of the threat modeling process, the security designer needs to understand the system in question completely. This entails understanding every component and its interconnections, defining usage scenarios, and identifying assumptions and dependencies. What we need is a system model that reveals the essential characteristics of the system.

Depending on the type of the system, it may be modeled using one of a number of different approaches. If the system in question is a software application, we can draw its *Data Flow Diagram* (DFD) which dissects the application into its functional components and indicates the flow of data into and out of the various parts of system components. The DFD approach makes it easier to identify threats than other system views because one can follow the adversary's data and commands as they are processed by the system, analyzing how they are parsed and acted on, as well as noting which assets they interact with [11]. One example of DFD is shown in our first case study: Software Defined Radio (see Figure 2).

Unfortunately, not all systems can be easily modeled using a DFD. Some systems are so complex that it is impossible to know all software components that may be present and thus no assumptions can be made about data flow paths. This is especially true for networked systems such as large-scale commodity clusters. Networked systems can be viewed via a *Network Model*, which allows analysts to examine communication between computers with different roles. When threat modeling a networked system, the first step is to identify the roles and function of each class of computers on the network. Then, the communication patterns between entities in different roles are mapped out. For example, application servers need to communicate with domain controllers and with user workstations. This mapping should pinpoint the protocols, ports, and traffic patterns used for these communications. An example of a network model can be found in our second case study: VisFlowConnect (see Figure 3).

2.2. Identifying Assets and Access Points

To properly utilize threat modeling, analysts must learn to think like the attacker. This is easier to do once the system is fully understood, hence the previous step in the threat modeling process was characterizing the system. Adversaries may be characterized in terms of their resources, access, risk tolerance, and objectives. The analyst should try to answer questions like:

- Who are my potential adversaries?
- What's their motivation, and what are their goals?
- How much inside information do they have?

Identifying assets and access points is a critical step in the threat modeling process. An *asset* is an abstract or concrete resource that a system must protect from misuse by an adversary. Assets can be tangible, such as processes and data, or more abstract concepts such as data consistency. It is impossible to have a threat without a corresponding asset because assets are essentially threat targets.

Access points are what the attacker is going to use to gain access to the assets. Examples of access points are open sockets, RPC interfaces, configuration files, hardware ports, and file system read/write. Related to access points, it is also important to determine the trust boundaries in the system. A trust boundary is a boundary across which there is a varied level of trust. For example, the network may form a trust boundary, as anyone can gain access to the Internet, but not everyone should have access to the enterprise system. Related to trust boundaries are trust levels. Trust levels indicate how much trust is required to access a portion of the system. For instance, if a user is an administrator, they are trusted to do more than normal users.

2.3. Identifying Threats

After the previous steps have been completed, it is time to think about the specific threats to the system. Threats may come from either inside or outside the system—from authorized users or from unauthorized users who masquerade as valid users or find ways to bypass security mechanisms. Threats can also come from human errors.

The goal of this step is to identify threats to the system using the information gathered so far. A *threat* is the adversary's goal, or what an adversary might try to do to a system [11]. Sometimes a threat is also described as the capability of an adversary to attack a system. In the context of threat modeling, the first definition is more suitable. it is often to start threat modeling with a list of known threats and vulnerabilities found in similar systems. Although working backward from known vulnerabilities typically yields common threats, system-specific threats require deeper analysis of the unique qualities of the system being modeled.

The best method for threat enumeration is to step through each of the system's assets, reviewing a list of attack goals for each asset. Assets and threats are closely correlated. A threat cannot exist without a target asset. Threats are typically prevented by applying some sort of protection to assets. The process of correlating threats to an asset involves creating adversary hypotheses. In threat modeling, a system model shows all security critical entities such as assets, access points, and communication channels. Threats can be identified by going through each of these security critical entities and creating threat hypotheses that violate confidentiality, integrity, or availability of the entity.

The output of threat identification process is a threat profile for a system, describing all the potential attacks, each of which needs to be mitigated or accepted. In general, threats can be classified into six classes based on their effect [11]:

- *Spoofing*- Using someone else's credentials to gain access to otherwise inaccessible assets.
- *Tampering*- Changing data to mount an attack.
- *Repudiation*- Occurs when a user denies performing an action, but the target of the action has no way to prove otherwise.
- *Information disclosure*- The disclosure of information to a user who does not have permission to see it.
- *Denial of service*- Reducing the ability of valid users to access resources.
- *Elevation of privilege*- Occurs when an unprivileged user gains privileged status.

When identifying a threat, it is helpful to think of various attacks in terms of the above classification. On the other hand, security threats are breaches of confidentiality, integrity, or availability. Thus, threats could also be classified by these properties. This classification is useful in security requirements when deciding on a mitigation mechanism of a specific threat. For example, unauthorized modification of data en route to component B from component A poses a tampering threat which violates the integrity property. To mitigate this threat, it might make sense to apply integrity mechanism such as Secure Hashing Algorithm-1 (SHA-1) on the data being transferred.

Identifying threats is only part of creating the system's threat profile. Threats must also be analyzed to determine whether the system is susceptible to them. Using attack trees is one way to accomplish this. To illustrate this process, consider the case in which an attacker wishes to decrypt a message traveling from machine A to machine B. At this point, the analyst must brainstorm to figure out all avenues the attacker may pursue in order to achieve this goal. These avenues become nodes under the original goal and become goals themselves that can be evaluated the same way. Note, attack trees cannot replace threat modeling process. Sheyner *et al.* [9] demonstrate the generation attack trees for limited, and somewhat artificial scenarios.

When defining a threat model, designers should be concerned not only with defining what attacks they are concerned with, but also those that are not a high priority. Risk assessment is performed to map each threat either into a mitigation mechanism or an assumption that it is not worth worrying about. At this point, the security requirements for the system can be defined.

3. Specifying Security Requirements

Security requirements provide foundations upon which the security of a computer system is built. Systems do not get developed without requirements; for a small, simple project perhaps requirements are not formally specified and documented, but still the developer would have some kind of requirements in his/her mind. Requirements specify *what* the system will do, whereas the following stages in software engineering and design define *how* it will work.

Security requirements are driven by security threats, whereas most requirements are based on higher level goals. While most requirements are stated in terms of what *must* happen, security requirements are often specified in terms of what *must not* be allowed to happen. For example, "The application shall not allow any customer to access the account information of any other customer." During this process, it is important to discover which threats are realistic and must be dealt with.

It is often impossible to mitigate every threat, and even if this could be done, it would almost certainly take place at the cost of decreased usability. Securing systems is about tradeoffs; often finding an ideal balance is quite a challenge. *Risk management* addresses this issue and is discussed in the next sub-section. Even if the security requirements process starts with a set of common requirements extracted from standards, it is still important to conduct a thorough risk management session to analyze the suitability of common requirements to the system. As mentioned previously, it can be the case that these common requirements are incomplete and security designers will need to specify additional requirements.

Once all potential threats have been analyzed and the threats that must be mitigated have been determined, the next task is to specify security requirements. A requirement is a statement of goals that must be satisfied. A security policy can be considered a set of security requirements. Mapping the security threats into the set of requirements and assumptions partially addresses the completeness of the system security. This mapping is used to justify the suitability of the proposed security mechanisms. In general, a requirements engineering process includes the following activi-

ties [5]: 1) Requirements elicitation; 2) Requirements analysis and negotiation; 3) Requirements validation. A threat model provides most of the information necessary for requirements elicitation. By converting a threat statement into a "shall not" requirement statement, it is possible to compile an initial set of security requirements. The details of requirements engineering are outside the scope of this paper; there are many excellent publications on the topic. Next, we demonstrate a mapping of a threat into a requirement.

Example. Let's say that a threat model for an electronic voting system has been defined. We will examine how one specific threat can be mapped to a security requirement. The threat in question is *"Attacker uses DoS attacks to reduce availability of the system"*. The attack tree reveals that this threat can be realized by either flooding the network interface or filling up the available disk space by writing spurious data to the audit log files [6].

During the risk assessment, we calculate the risk of the threat in several categories. For each category, assign a risk value from 1 to 10 calculated as a multiple of criticality by likelihood.

Number of affected users	10
Damage potential	6
Level of skills needed	6
Cost of attack	8
Reproducibility	8
Discoverability	10
Total Risk	**8**

We decide to mitigate this threat because the total risk factor is fairly high. The requirement for mitigating this threat could be: *"The system shall not allow any user to successfully use DoS attack to reduce availability of the system"*. To satisfy this requirement, the following mitigation measures are proposed: a) Use a firewall to drop certain IP packets; b) Restrict resources used by anonymous users.

3.1. Risk Management

Despite the best efforts of security researchers, it is impossible to guarantee 100% security. However, we can work toward 100% risk acceptance. A good security system strikes a balance between what is possible and what is acceptable via the risk management process. Risk management consists of risk assessment, risk reduction, and risk acceptance.

To assess the risk of identified threats, the threats must be prioritized. The simplest way to prioritize threats is by using two factors: damage and likelihood. First, calculate the overall risk factor for each threat, and then sort the threat list by decreasing order of risk. Threats can be addressed starting at the top of the list. There are four possible ways to manage a risk:

Accept the risk - The risk is so low and so costly to mitigate that it is worth accepting.

Transfer the risk - Transfer the risk to somebody else via insurance, warnings etc.

Remove the risk - Remove the system component or feature associated with the risk if the feature is not worth the risk.

Mitigate the risk - Reduce the risk with countermeasures.

The threats selected for mitigation must be addressed by some countermeasure. Designers should ensure that security does not become more expensive than it is worth. Security measures should be employed only up to the point where the cost to implement them does not exceed the expected risk. Failure to make this judgment correctly can easily lead to a situation where no risk is judged acceptable, and thus no acceptable system can be designed.

4. Case Studies

In this section we demonstrate how the threat modeling process can be applied to different types of systems through specific examples. The first example is a host-based software application called *GNU Software Defined Radio*, the second is a network-based traffic monitoring system called *VisFlowConnect*, and the last is a cluster security monitoring tool called *NVisionCC*. In each case study, we characterize the system with an appropriate system model, identify main assets and access points, and finally identify threats. We do not provide security requirements for these systems at this time, it is a work in progress.

4.1. Software-Defined Radio

Software Defined Radio (SDR) implements radio functionality in software that was previously provided in hardware, allowing the radio to function as any type of radio device such as cellular phone, GPS receiver, or packet radio. Some of the advantages of SDR are: reconfigurable handsets and infrastructure equipment, general-purpose hardware, over-the-air software updates, easy migration of networks from one generation to another, and seamless network access across various geographies. We were involved in a threat analysis of SDR [3]. A data flow approach was used to understand the components and interconnections of the software before identifying its security threats.

GNU SDR [1] is a open-source software platform that provides radio operating environment and digital signal processing library. It is organized around the principle of constructing a graph that describes the data flow of the radio system. The vertices of the graph are signal processing blocks that perform various mathematical manipulations on the input signal stream, e.g. modulation, filtering, mixing

etc. The edges connecting adjacent signal processing blocks contain shared buffers that are used to temporary store the signal. The radio system starts its operation when the flow graph is handed off to the runtime system for execution. Figure 2 shows a flow graph of an FM receiver.

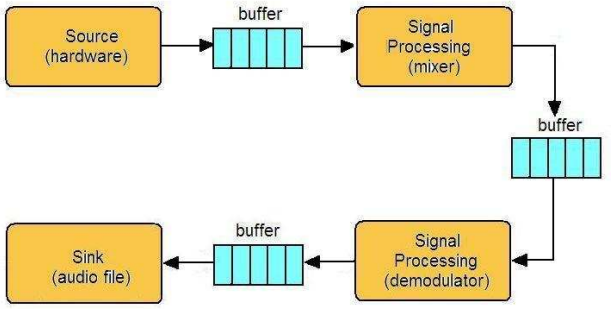

Figure 2. Data flow graph of an FM receiver.

The main assets of GNU SDR that attackers may target are:

- Flow graph
- Scheduler
- Signal processing library
- Shared buffers
- Signal stream

Based on these assets and access entry points, we identified the threats of GNU SDR. For each threat, we name the threat, provide its classification, and describe its effects and consequences.

- Any entity may construct the flow graph in the current implementation of the system (Tampering). This allows an adversary to compose an eavesdropping or jamming device.
- Incompatible software modules may be assembled in the flow graph without adherence to regional regulations and hardware specifications (DoS). This may render the device inoperable.
- Insecure over-the-air software download (Tampering, Disclosure). Software may be illegally modified or an adversary may supply its own software.
- Downloaded software may contain buffer overflow problem (Tampering). Even if software was supplied by a certified vendor over a secure communication channel, it may have faulty functionality. This allows an adversary to modify the signal stream in shared buffers at will or execute a malicious code.
- Signal processing blocks may access other module's data (Tampering, Disclosure). The scheduler executes signal processing blocks in the flow graph sequentially, one after another, as a single process. This again allows an adversary unhampered access to all shared buffers in the flow graph.

There are other threats such as unauthorized use of network services and unauthorized login into a radio device that concern SDR in general but outside the scope of GNU SDR software. Based on our risk management of the threat profile, we recommend the following security measures for GNU SDR:

- Encryption and integrity check for over-the-air software download.
- Verify that downloaded software is supplied by a certified vendor.
- Design a trusted configuration module responsible for a flow graph construction.
- Design a policy-driven configuration framework to automate the mapping of user request into a functional flow graph. For example, when a user requests an FM receiver, the system is knowledgeable enough to construct the graph shown in Figure 2.
- Provide fault domain isolation for signal processing blocks so that each has access only to its own memory area.
- Provide a buffer overflow prevention mechanism, perhaps as a compiler extension or kernel extension.

4.2. Network Traffic Monitoring Tool- VisFlowConnect

VisFlowConnect [14] is a visualization tool designed to enhance the ability of a network administrator to detect and investigate anomalous traffic. Also termed as a passive monitoring tool, VisFlowConnect visualizes the network traffic between the enterprise network and the Internet. It is capable of filtering traffic to display only traffic with certain attributes and indicate traffic dynamics over time by animating flows with a user adjustable time window. VisFlowConnect relies on NetFlow [12] data as its input data source. Each record in NetFlow log files contains a variety of information about a network flow such as source/destination IP address and port number, number of bytes and packets transferred, protocol used, and start/end time of the transmission.

Figure 3 shows the architecture of NetFlows deployment at NCSA. NetFlow collectors consume flows from multiple sources, perform data reduction through filtering and aggregation, and store flow information in flat files ready for further processing. Then flow datagrams are converted into

uniform NCSA format, processed by flow processors, and fed as a data source to VisFlowConnect tool.

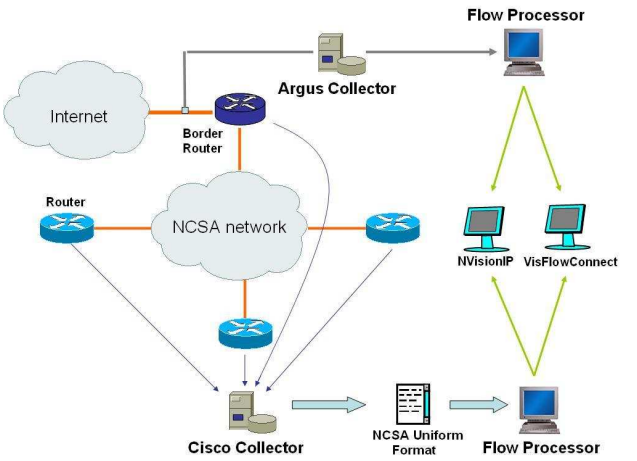

Figure 3. Architecture of NetFlows deployment at NCSA.

Although VisFlowConnect in itself is a security monitoring tool, it is essential to identify its threats. Threat modeling of VisFlowConnect helps understand its limitations in detecting DoS attacks against the monitored network, and ensure that the tool does not introduce new vulnerabilities to the system or inadvertently makes attacks on the system easier. We use a network model for threat modeling VisFlowConnect. Based on our understanding of the system architecture, we identify the following security-critical assets:

- NetFlow collectors
- Flow processor
- Format converter
- NetFlow datagrams
- Data buckets on flow processor
- CPU cycles on collectors
- Communication channels between routers, collectors, and flow processor
- Software library used by VisFlowConnect

The following threats arise in relation to these security-critical assets:

- Routers may not log network flows accurately. This could happen if the routers housing VisFlowConnect are compromised.
- NetFlow logs on routers may be tampered with. Log files modified before reaching the collectors.
- Confidentiality and integrity of flow datagrams compromised while en route between routers, collectors, and flow processor.
- Flow datagrams delayed, replayed, or missing.
- Any of the software components that handle NetFlow datagrams may be faulty (buffer overflow) or malicious.
- An adversary might adapt its attack pattern to mimic normal network traffic.

4.3. Cluster Security Monitoring Tool- NVisionCC

Clusters are vulnerable to new types of attacks that are not possible with individual hosts. For example, a brute-force attempt at guessing passwords can be distributed across the cluster to enable the attack to succeed without activating host-based security mechanisms checking for frequently failed logins. By treating a cluster as a single indivisible entity instead of a collection of individual nodes, it is possible to effectively monitor the security of a cluster and catch this type of attack. NVisionCC is an active monitoring tool for cluster security that monitors processes across cluster nodes and raises alerts when deviations from a predefined profile of expected processes are noted [4, 13].

The process monitoring architecture consists of three separate components. The *information collector* gathers process data from cluster nodes and stores it in a central database as shown in Figure 4. The *data analyzer* retrieves the data from this database and searches for deviations from the user-defined profiles for each type of node such as a head node, compute node, storage node, etc. The *visualization client* combines the data from the database with the results from the data analyzer to provide a user with a visual representation of the cluster state. NVisionCC is implemented as a plugin to the Clumon cluster monitoring tool. The information collector uses Clumon's process monitoring infrastructure.

Clusters are typically the highest valued asset within an organization's environment and as such should receive priority security protection. Although NVisionCC is a security mechanism for clusters, it may have its own security vulnerabilities, or even worse, it may expose a cluster to new threats. As with any system, security analysis of NVisionCC must begin with a threat model. We identify the following security-critical assets for NVisionCC:

- PCP daemon
- Information collector
- Data analyzer
- Visualization client

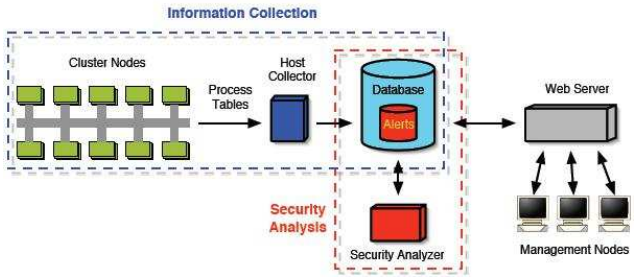

Figure 4. Architecture of NVisionCC.

- Process data
- Attack alerts
- Normal state profiles
- Communication channels between collector, analyzer, and visualizer
- Software library used by NVisionCC

The following threats arise in relation to these security-critical assets:

- A malicious process may masquerade as a legitimate process on a cluster node.
- PCP daemon could become altered, compromised.
- Process data on cluster nodes may be tampered with.
- Confidentiality and integrity of process data compromised while en route to the information collector or the central database.
- Process data delayed, replayed, or missing.
- The integrity of attack alerts sent from the analyzer could be compromised.
- If the attack is completed fast enough between PCP polls, it can escape detection.
- The attacker may learn the PCP polling pattern and try to subvert detection by spreading his/her activity over time.

5. Conclusions

In this paper we propose threat modeling as an essential foundation for defining security requirements of computer systems. Without identifying threats, it is impossible to provide assurance for the system and justify security measures taken. We have presented our view on best practices for the threat modeling for both software applications and complex systems.

In the future, we plan to define security requirements based on threat modeling for several security tools developed at NCSA including VisFlowConnect, NVisionCC, and NVisionIP. Ultimately, we will design and implement the security mechanisms necessary for these systems. These projects will help us to better understand and explore the interplay between threat modeling and requirements definition for systems which have already been deployed.

References

[1] GNU Software Radio project. http://www.gnu.org/software/gnuradio/.

[2] (CCEB) Common Criteria Editorial Board. Common Criteria for Information Technology Security Evaluations. *Report*, 1998.

[3] R. Hill, S. Myagmar, and R. Campbell. Threat Analysis of GNU Software Radio. In *Proc. of World Wireless Congress (WWC'05)*, Palo Alto, CA, May 2005.

[4] G. A. Koenig, X. Meng, A. J. Lee, M. Treaster, N. Kiyanclar, and W. Yurcik. Cluster Security with NVisionCC: Process Monitoring by Leveraging Emergent Properties. In *Proc. of IEEE Cluster Computing and Grid (CCGrid '05)*, May 2005.

[5] G. Kotonya and I. Sommerville. *Requirements Engineering: Processes and Techniques*. John Wiley & Sons, 1998.

[6] G. Obradovic. Threat Modeling and Data Sensitivity Classifi cation for Information Security Risk Analysis. *Presentation at Data Protection '03*, 2003.

[7] B. Schneier. Why Cryptography is Harder than it Looks. *Electronic Article*, 1997.

[8] B. Schneier. *Secrets & Lies*. John Wiley & Sons, Inc., 2000.

[9] O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J. Wing. Automated Generation and Analysis of Attack Graphs. In *Proc. of IEEE Symposium on Security and Privacy*, April 2002.

[10] O. Sheyner and J. Wing. Tools for Generating and Analyzing attack Graphs. In *Proc. of Formal Methods for Components and Objects*, pages 344–371, 2004.

[11] F. Swiderski and W. Snyder. *Threat Modeling*. Microsoft Press, 2004.

[12] C. Systems. NetFlow Services and Applications. *Technical Report*, 1999.

[13] M. Treaster, G. A. Koenig, X. Meng, and W. Yurcik. Detection of Privilege Escalation for Linux Cluster Security. In *Proc. of 6th LCI International Conference on Linux Clusters*, April 2005.

[14] X. Yin, W. Yurcik, M. Treaster, Y. Li, and K. Lakkaraju. VisFlowConnect: NetFlow Visualizations of Link Relationships for Security Situational Awareness. In *Proc. of ACM Workshop on Visualization and Data Mining for Computer Security (VizSEC '04)*, Washington DC, 2004.