Composable Tools
For Network Discovery and Security Analysis

Giovanni Vigna Fredrik Valeur Jingyu Zhou Richard A. Kemmerer

Reliable Software Group
Department of Computer Science
University of California Santa Barbara
[vigna,fredrik,jzhou,kemm]@cs.ucsb.edu

Abstract

Security analysis should take advantage of a reliable knowledge base that contains semantically-rich information about a protected network. This knowledge is provided by network mapping tools. These tools rely on models to represent the entities of interest, and they leverage off network discovery techniques to populate the model structure with the data that is pertinent to a specific target network. Unfortunately, existing tools rely on incomplete data models. Networks are complex systems and most approaches oversimplify their target models in an effort to limit the problem space. In addition, the techniques used to populate the models are limited in scope and are difficult to extend.

This paper presents NetMap, a security tool for network modeling, discovery, and analysis. NetMap relies on a comprehensive network model that is not limited to a specific network level; it integrates network information throughout the layers. The model contains information about topology, infrastructure, and deployed services. In addition, the relationships among different entities in different layers of the model are made explicit. The modeled information is managed by using a suite of composable network tools that can determine various aspects of network configurations through scanning techniques and heuristics. Tools in the suite are responsible for a single, well-defined task. Each tool has an abstract specification of the input, the output, the type of processing, and the requirements for carrying out a task. Tool descriptions are expressed in a Network Tool Language. The tool descriptions are then stored in a database. By using the network model and the tool descriptions, NetMap is able to automatically determine which tools are needed to perform a particular complex task and how the tools should be scheduled to obtain the requested results.

Keywords: Network Security, Network Modeling and Analysis, Network Discovery and Validation.

1. Introduction

Network security is achieved by composing the functionality of a number of security applications, such as firewalls and intrusion detection systems. Deploying and configuring security applications requires an in-depth knowledge of the network to be protected. In addition, continuous monitoring of both the network and the configuration of the security applications is the basis for determining the current network security posture.

Unfortunately, knowledge about the network being protected often exists only in the "mind" of the network administrator, and this knowledge is obtained by using a number of tools, each of which can only provide a subset of the information about the protected network. For example, the information about the services active on a host could be determined by scanning the ports of the host. In addition, the results obtained from the execution of one tool are often used as the basis for additional analysis and possibly as input for the execution of other tools. In the previous example, once the open ports have been determined, banner-grabbing tools can help to determine the type and version of the server applications. The coordination of tool executions and the composition of their results is usually a human-intensive task. This is the case even when *ad hoc* scripts and procedures developed by network administrators through years of experience in integrating the results of network monitoring and analysis are available.

This paper presents NetMap, a novel approach that provides support for automated network discovery and security analysis. NetMap is centered around a model of both the network to be analyzed and the tools to be used for analysis.

The network model has been designed by taking into account the models used by existing network management and vulnerability scanning tools. The model is not limited to a specific network level; it integrates network informa-

tion throughout the layers. The model contains information about topology, infrastructure, and deployed services. In addition, the security-relevant relationships between different entities in different layers of the model are made explicit. For example, the model includes trust relationships between clients and servers for specific services, as well as relationships between services and configuration objects (e.g., files) used to define the application behavior. The network model is implemented as a database management system, called NetDB.

A tool model supports the abstract description of a suite of network discovery and scanning tools using a Network Tool Language (NTL). Each tool in the suite is responsible for a single, well-defined task and has a specification of the input, the output, the type of processing, and the requirements for carrying out a task. The tool descriptions are stored in a tool repository, called the Network Tool Database (NTDB).

NetMap allows a network administrator to specify high-level discovery/analysis tasks in a query language, called NetScript. Tasks range from pure network discovery, to the validation of existing information, to vulnerability scanning. Given a task description, a Query Processor component uses the tool descriptions to determine which tools are needed to perform a particular complex task, what their schedule should be, and how the results should be inserted into an instance of the network model that represents the protected network.

The remainder of this paper is structured as follows. Section 2 discusses related work on network models and network analysis tools and presents an overview of the NetMap approach. Section 3 describes the network model. Section 4 presents the concept of composable network tools. Section 5 discusses issues related to network discovery and security analysis. Section 6 presents an evaluation of NetMap's performance. Finally, Section 7 describes the current status of the NetMap system, draws some conclusions, and outlines future work.

2. Related Work

Currently, networks are monitored, maintained, and diagnosed using tools that rely on network protocols like the Internet Control Message Protocol (ICMP) [10] and the Simple Network Management Protocol (SNMP) [2]. Examples of these tools are HP OpenView [6], Scotty [11], Brother [1], and Fremont [12]. These tools support network discovery tasks and provide a means to remotely query and control network devices, such as routers and hosts.

Network management tools have proved to be effective in determining network configuration problems and in helping security analysts. However, their data model and the type of information they gather is not sufficient to determine and verify the security posture of a protected network. Thus, network security analysts use vulnerability scanning tools in addition to network management tools. Vulnerability scanning tools automatically perform checks on the hosts of a subnetwork looking for vulnerable applications, misconfigured services, and flawed operating system versions. Examples of these tools are Nessus [8], Nmap [4], and ISS's Internet Scanner [7]. These tools provide different types of functionality, use different means to retrieve information about a network, and store information in different formats. Table 1 summarizes the characteristics of several popular network and security analysis tools. The table shows, for each tool, the type of functionality provided (node discovery, topology discovery, service mapping, operating system fingerprinting, and node management), and the type of storage used for the information gathered (data structures in memory, text files, or databases).

The tools described above provide many useful functionalities but suffer from four main limitations:

1. *They are limited in scope.* Most of the tools address one single problem (e.g., Nmap provides only scanning capabilities). Different analysis domains, such as routing and application-level service configuration, are not analyzed in an integrated way.

2. *They do not rely on a well-defined, shared network model.* Some tools do not model and store persistent data at all, others use text files that are mostly unstructured. A few rely on database management systems, but the corresponding database schemas are designed for the specific tool only; they do not cover features not considered by the tool. In addition, these tools do not agree on a shared model. This makes it hard to combine the results from one tool with another. Even though there are ongoing efforts to standardize a network management model [3], the proposed standard does not take into account the application-level characteristics of a network, which are paramount in determining the security configuration of services.

3. *They are not flexible.* In most cases, it is impossible or very hard to add new functionality and analysis techniques to an existing tool. The recent vulnerabilities discovered in a number of SNMP implementations [5] have brought this problem to the forefront. In order to cope with the increasing number of attacks targeting SNMP agents, the agents have often been disabled, which effectively prevents SNMP-based network tools from working properly. Even though the desired information is accessible by other means (e.g., by remote execution of shell scripts), the existing tools cannot be easily modified to take advantage of these alternative sources of information. In addition, composing and

Product	Description	Functionality					Storage
		Node Discovery	Topology Discovery	Service Mapping	OS Fingerprint	Node Management	
Nmap	Port scanning tool	√		√	√		Memory
Nessus	Vulnerability scanning tool	√		√	√		Memory
Fremont	Topology discovery tool	√	√				Memory, text files
Big Brother	Network monitor					√	Text files
Scotty	Network Management Tool	√	√	√		√	Memory, text files
OpenView	Network Management Tool	√	√			√	Database

Table 1. Characteristics of existing network analysis tools.

integrating different tools requires the development of *ad hoc* procedures.

4. *There is no automated support for tool composition.* Given a network analysis or monitoring task, there is no automated support to determine what tools could be used to carry out the task or how different tools should be composed.

NetMap is a new approach that overcomes the limits listed above. NetMap's goal is to provide a network analysis tool that supports network discovery and analysis over a wide range of network characteristics. NetMap relies on a well-defined network reference model to represent both the entities of a protected network and a suite of network analysis tools. Network discovery tools are used to populate the model structure with the data that is pertinent to a specific target network, and then security analysis is performed on the collected data. Unlike any other network security or network management tool, the approach presented in this paper does not rely on a monolithic tool suite or a fixed set of techniques. The NetMap approach relies on composable network tools. NetMap maintains a tool database containing a toolset composed of specially built tools, COTS components, or specific tool features (e.g., the TCP portscanning functionality of Nmap). Each of the tools in the toolset is responsible for a single well-defined task (e.g., determining if a host in a network is up or down) and is associated with a specification of the input, the output, the type of processing, and the requirements for carrying out a task. The tool description is expressed in a Network Tool Language. The tool descriptions are then stored in the Network Tool Database.

Whenever data has to be retrieved to populate the network model or to verify its contents, a Query Processor component automatically determines:

- what information is needed;
- which tools can be used to obtain or verify the information; and
- how to compose the inputs and outputs of different tools to obtain the result.

The resulting system is flexible, customizable, extensible, and can easily integrate off-the-shelf tools. In addition, it provides automated support for the execution of complex tasks that require the results obtained from several different tools. Figure 1 shows the high-level architecture of the system. Existing network tools are described using NTL specifications. Tool specifications are stored in the NTDB. The Network Security Administrator browses the network information contained in the NetDB and may request the execution of a network discovery operation by issuing a NetScript query. The query is sent to the Query Processor component, which determines a suitable set of tools to perform the requested task on the basis of the information stored in both the NetDB and the NTDB. The tools are scheduled for execution, the actual tools are invoked, and eventually the results are stored in the NetDB for further analysis. The following sections detail the main components of the NetMap architecture.

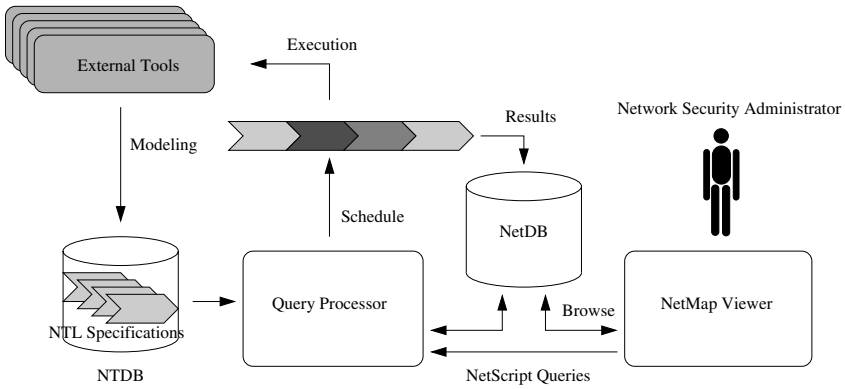

Figure 1. NetMap high-level architecture.

3. The Network Model

The network model is an entity-relationship description of a network. It describes both the topology and the service structure of the network. Figure 2 presents a simplified schema for the model.

The network topology is a description of the constituent components of the network and how they are connected. The network model defines entities, such as interfaces, nodes, and links, to describe elements of the network, and uses relationships to determine how the elements are connected to each other. Each topology element has a rich set of attributes that defines the characteristics of the element. For example, the node element is characterized by its type (e.g., a router or a workstation), the processor architecture, type, and speed, the manufacturer, the amount of memory and disk storage available, its geographical location (e.g., building and room number), and so on. This part of the model represents the "blueprint" of a network.

The network model is not limited to network topology; it also contains a description of the service structure provided by the hosts of a network. This includes what operating systems are installed on the different hosts, and what services are available. Examples of these services are the Network File System (NFS), the Network Information System (NIS), Secure Shell, FTP, and "r" services. The model contains a characterization of each service in terms of the network/transport protocol(s) used, the access model (e.g., request/reply), the type of authentication (e.g., address-based, password-based, token-based, or certificate-based), and the level of traffic protection (e.g., encrypted or not). In addition, the model explicitly represents the relationships between the different entities. For example, the model includes trust relationships between service clients and servers, as well as relationships between services and configuration objects (e.g., files) used to define program behavior. This structure allows one to determine the implicit impact of an attack with respect to the whole network. For example, suppose that a host-based attack that allows an unauthorized user to write to a root-owned file (e.g., */etc/exports*) is detected. The model contains the information that relates the target file to a specific service (in this case, the Network File System). Analysis based on the model can determine the overall impact of the attack. For example, suppose that client hosts use NFS to mount users' home directories. The NFS service could be used to mount modified versions of the users' environments extending the compromise to many user accounts. In this case, by making the relationship between the client and the server explicit it is possible to understand that a simple attack is actually affecting the security of all of the users of the network.

The model is implemented using a relational database. The model explicitly addresses three different levels: structure, view, and status. At the structure level the model represents those objects that have a relatively long lifetime, such as topology and services. At the status level the model represents information related to the current status of the network, such as network statistics. At the view level the model provides different metaphors to present the information contained in the model to the users (or to applications). Currently, two prototype views have been implemented; one is based on the tkined system [9], and a second is accessible through a Web interface. Both views allow the Network Security Administrator to browse the NetDB, update the contained information, and issue NetScript queries to the Query Processor component.

4. Composable Network Tools

Network discovery and analysis is done by building new tools or using tools that already exist and combining them to achieve the desired results. The advantage of using existing tools is that it requires less work to implement the mapping and analysis procedures.

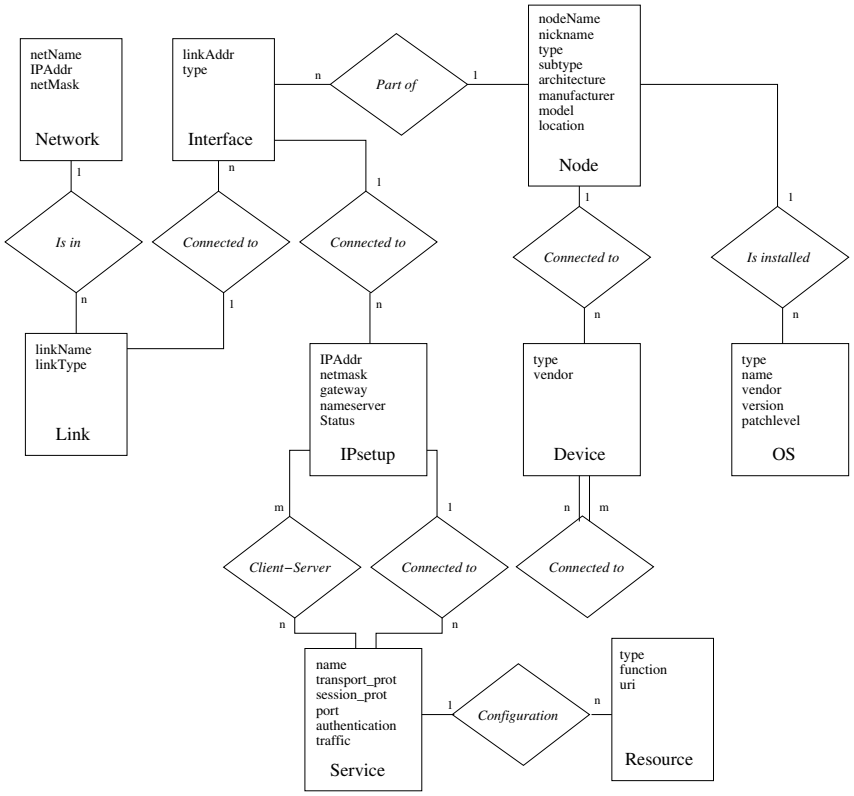

Figure 2. Entity-relationship schema for the network model.

The NetMap philosophy is to perform each part of the overall network discovery/analysis task with the tool best suited for the job. This works best if a tool performs one specific task instead of implementing many different functionalities as a monolithic application. A tool that is able to perform many tasks should at least have parameters that can limit the operation of the tool to exactly what is needed. An example of a tool like this is Nmap [4]. Nmap performs ping scans, port scans, OS fingerprinting, and RPC scans. Nmap can be finely tuned to suit the specific needs of each query.

NetMap provides a way to describe the characteristics of network tools by writing specifications in the Network Tool Language. These specifications serve as the basis for determining which tools to run and how to compose their input and output. Each tool can have different costs associated with it. For instance, a cost could represent efficiency, confidence, or network bandwidth usage. The purpose of the cost metrics is to provide support for selecting the most appropriate tools to answer a particular query.

When NetMap is given a query in the NetScript language, the Query Processor determines all the possible tool schedules that satisfy the query. These schedules are constructed so that they satisfy all the dependencies in the tool descriptions. If more than one schedule can answer the query, the schedule that optimizes the desired cost metrics is selected. The selected schedule is then run. Note that the schedule that optimizes efficiency is not likely to be the same as the one that optimizes confidence. Therefore, the user queries must also specify what cost metrics are most important.

4.1. Representing Model Entities

NetScript and NTL must agree on a common way to refer to entities in the network model. A NetScript query uses these references to represent a desired value. NTL uses references to entities and their attributes to specify the input required by a tool and/or the tool's output.

The set of attributes of interest is specified by using a path and then organizing the paths into trees. A path is a list of identifiers separated by dots, where the first identifier is called the root, intermediate identifiers are relation names, and the last identifier is an attribute name. If several paths start with a common subsequence of identifiers they can be combined into a tree. The tree is formed by concatenating the common subsequence with a comma separated list of the remainders enclosed in parentheses, where a remainder is the path with the common subsequence removed. For instance, the paths `iface.mac` and `iface.type` can be

```
tool ping {
  ipsetup s;
  input s.ipaddress;
  output s.status;

  efficiency = 2;
  confidence = 9;

  code{
    nargs -n 1 -i "nmap -sP {} |
      if grep 'appears to be up.' >/dev/null ; then
        echo 1;
      else
        echo 0;
      fi"
  }
}

tool nmap_portscan {
  ipsetup s;

  input s.ipaddress;
  output s.services*-.(port,transport_prot);

  efficiency = 5;
  confidence = 6;

  // The following input assertion is used
  // to limit the space of a portscan to local networks
  input_assertion ipsetup.ipaddress:InIpRange(128.111.*.*);

  // The tool is only able to scan TCP ports
  output_assertion service.transport_prot:Equals(TCP);

  code{
    ssh root@host ". ./nminit;nm_portscan"
  }
}
```

Figure 3. Example of the tool definition syntax.

combined into the tree `iface.(mac,type)`.

Intermediate identifiers in the tree can be marked with set qualifiers. The valid set qualifiers are "`*`" for complete set, "`*-`" for subset, and "`*+`" for superset. The qualifier should occur directly after an identifier in the path. The set qualifiers are used to map entities and attributes of interest into sets of entities in the Network Model. For example, consider `node.nodename` and `node*.nodename`. The first case refers to one node's nodename, while the second case refers to the set of nodenames for all nodes.

4.2. The Network Tool Language

NetMap tools are described using the Network Tool Language. A tool description starts with the keyword "`tool`". This is followed by the name of the tool and a tool description body enclosed in curly brackets. See Figure 3 for two examples.

The tool description body consists of optional variable declarations, an optional input definition, an output definition, optional cost specifications, optional assertions, and a code block. The elements are separated with a "`;`".

The input and output definitions are the tool description's most important parts. The Query Processor needs to know about a tool's input and output to resolve tool dependencies. Before a tool can be run, all the input data it needs must be present. If some input data is missing a different tool must be run first to provide the required data.

Input and output definitions have similar syntax. They start with the keyword "`input`" or "`output`" followed by a tree that contains all the attributes that are to be defined. The root element is either the name of an entity or a declared variable.

In order to specify a relation between the input parameters and the output parameters, NTL supports the declaration of variables that can be used as root elements in both the input and output definition. The declaration starts with the type of the entity to be declared followed by a variable name. For example, in both example descriptions in Figure 3, a variable of type `ipsetup` is declared and then used in both the input and output definition.

The syntax for cost specifications is *costname* "`=`" *value*, where *value* is relative to a specified range. Different cost metrics may be associated with different ranges.

The syntax for a code block starts with the keyword "`code`" followed by the code to be executed enclosed in curly brackets. The code represents the set of actions to be executed to carry out the tool's task.

There are tool assertions for both input data and for output data. An assertion is only in effect when the tool in question is run. The input assertion, introduced by the "`input_assertion`" keyword, is used to require that the tool is run only using input entities that have some special attributes. For example, a tool that checks a particular web server feature needs a web server to be present on the target host. Some tools can also be dependent on the target computer running a specific operating system. The output assertion, introduced by the keyword "`output_assertion`," is used to filter unwanted excess data from the output of a tool. Output assertions are also used by the Query Processor in the scheduling process. If some tools are only capable of scanning a limited value set, then the scheduler can combine the tools in order to cover the whole value domain. The Query Processor does not support this feature in the current implementation. For both types of assertions, the initial keyword is followed by an attribute reference, a "`:`", and the assertion. The attribute reference is of the form *entity name*`.`*attribute name*. The format of the constraint specification is dependent on the type of assertion.

In order for a tool's code to gain access to the assertions and their constraint specifications, assertion hooks are provided. An assertion hook can appear anywhere within a code block. It starts with the character sequence "`#ASRT`" and ends at the first following "`#`". The body of the hook is composed of tokens separated by "`:`". The tokens specify

```
query nodescan(iprange) {
  result ipsetup*.ipaddress;

  assertion ipsetup.ipaddress:inIprange(iprange);

  confidence 2;
  efficency 1;

  code {
    <.. Code to process the result ..>
  }
}

query portscan(iprange, portrange) {
  result ipsetup*.(ipaddress,
                   services*.(port,
                              transport_prot,
                              name));

  assertion ipsetup.ipaddress:InIpRange(iprange);
  assertion service.port:InRange(portrange);

  code {
    <.. Code to process the result ..>
  }
}
```

Figure 4. Example of the query syntax.

the assertion of interest, the attribute of interest, and other parameters that allow a tool to use the constraint information at execution time.

The `ping` tool in Figure 3 declares a variable of type `ipsetup`, which is used as the root in the input and output definitions. The tool takes an IP address as input and outputs a status flag. A non-null value for this flag means that the host is answering ICMP echo messages. A confidence cost of 2 and efficiency cost of 9 is specified. The code runs Nmap in ping scan mode.

The `nmap_portscan` tool in Figure 3 takes an IP address as input and returns a list of ⟨`port`,`transport_prot`⟩ tuples representing the services related to the IP address. The code block specifies that a command should be executed on a remote host to do the scanning.

4.3. The NetScript Query Language

Queries are a way of issuing commands to NetMap to start the discovery of the parts of the network that one is interested in. A NetScript query specifies the network attributes of interest, the range of values they can have, and what to do with the result.

See Figure 4 for two examples of the NetScript syntax. A query definition starts with the keyword "`query`" followed by the name of the query and a comma separated list of parameter names in parenthesis. This is followed by the body of the query in curly brackets.

The query body consists of a result specification, assertions, cost weights, and a code block. The result specification and the code block have the same format as in the NTL tool descriptions. The result specification is the only mandatory part. Assertions start with the keyword "`assertion`". The rest of the syntax is the same as for NTL tool assertions. The syntax of cost weights are the cost-name, a whitespace, and a weight. The statement is terminated by a "`;`".

The result specification identifies which attributes are of interest. The assertions set a limit on the value the attributes can have. Assertions can, for instance, constrain a query to a subnet. The cost weights state how important each cost is when deciding which tools to use. NetMap currently supports two different classes of costs depending on how the total cost is calculated. One class uses the sum of all costs as the total, while the other uses the minimum value. The sum type is appropriate for an efficiency cost, while the minimum type would be used for a confidence cost. The code block is run after the query is finished. The purpose of the code block is to process the result.

The `nodescan` query in Figure 4 takes one parameter as input. The input definition asks for a range of IP addresses. The assertion limits the range of IP addresses that is scanned to the parameter passed to the query. The cost statements specify that confidence is twice as important as efficiency. The second example query asks for a range of IP addresses and the related services' port numbers, transport protocols, and service names. The two assertions limit the IP addresses and the ports that are scanned to the parameters passed to the query.

5. Managing Network Information

After having successfully run the tools, the Query Processor stores the query result in the NetDB database. One of the problems that might occur is that the data received from the tools is inconsistent and/or incomplete. The Query Processor uses a normalization procedure to generate a consistent view of the network from the current content of the database.

5.1. Resolving Inconsistencies

The most common inconsistency problem is the handling of so-called *ghost entries*. A ghost entry is present when more than one of the stored instances represent the same network object. This often happens when tools return instances with few or no attributes. In this case, the Query Processor cannot immediately tell if these instances were previously stored or not; therefore, ambiguities must be resolved by post processing the data.

Constraints offer a way to determine if two entity instances represent the same network object or not. A unique constraint on an attribute means that the attribute uniquely identifies the entity instance, similar to keys in a database

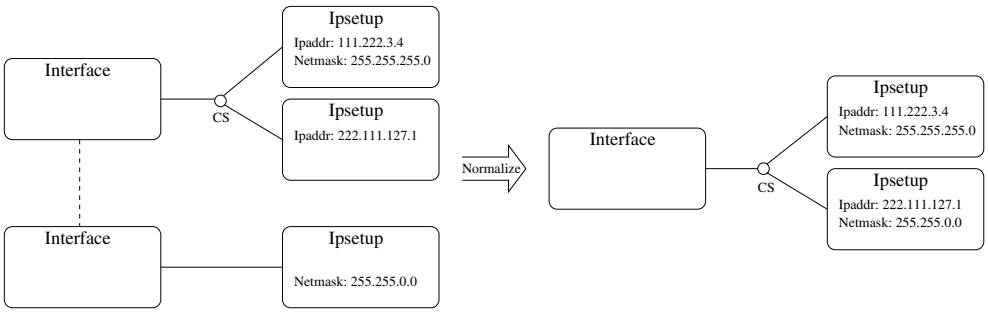

Figure 5. Example of the complete set constraint.

table. Note that the Query Processor allows the NetDB to be in a semi-inconsistent state, where more than one entity instance may have the same unique attribute value. This inconsistency is resolved during the normalization of the database, when all instances that have the same unique attribute value are merged.

The cardinality constraints on the relations in the network model can also be used when resolving ghost entries. Consider a relation that has a 1:N constraint. If the data stored in the NetDB actually implements an M:N relation, then the Query Processor can infer that all the entity instances on the left side of the relation are ghost entries. This inconsistency can then be resolved by merging all instances on the left side of the relation.

Another useful constraint is the "complete set" constraint. A relation instance is marked as a complete set if it is known that no more entity instances can take part in that relation. If other related entities exist, then they are ghost entities and should be merged with the complete set. As an example of the use of a complete set constraint, consider the `interface` and `ipsetup` entities from the NetDB schema of Figure 2. Figure 5 shows a graphical representation of an example input to the normalization algorithm and the result. The dashed line between the two interface elements symbolizes that the two interface instances shown are the same. The "CS" next to two of the relations denotes that the relation is a complete set. The normalization algorithm detects that there exists one `ipsetup` instance that is not part of the complete set. Because of the complete set constraint, the `ipsetup` entity must be a ghost entry of one of the `ipsetups` in the complete set. The `netmask` attribute of the ghost entry and the first `ipsetup` in the complete set differ. This means they cannot represent the same object. The only possible solution is that the ghost entry and the second `ipsetup` are the same and should be merged. The result of the algorithm is shown in the right side of Figure 5.

5.2. Network Security Analysis

After the NetDB database is populated with up-to-date network information, a comprehensive security analysis can be performed. The output of the analysis may either be a report of the current state of the network or configuration data to be used with some security component, such as a firewall or an intrusion detection system.

Currently, two prototype analyzers have been developed. The first, a firewall configurator, uses the client-server relationship expressed in the network model to create a list of valid clients for each service. The list can be used by the firewall to block unauthorized clients from accessing sensitive services. Even if a malicious user were able to change the access control list of the service itself, he would not be able to gain any access, since the firewall would block any connection attempt.

The second analyzer lists all the hosts in the network with a given operating system that have a specific service installed. This information is used when a network administrator needs to decide which hosts are affected by a new security vulnerability and need patching. Without a database of all installed services in the network, this information would have to be collected by some *ad hoc* scanning tool. The construction of this tool would be time consuming, and the results would likely be error-prone due to the *ad hoc* nature of the tool.

6. Evaluation

NetMap's functionality and performance have been tested on both simulated and real networks. The real networks that have been scanned are subnets in the Computer Science Department at UCSB. The tests on these real networks were performed to check whether NetMap is able to map and analyze a network correctly. The tests also gave information about how long the discovery process takes. The tests performed on the simulated network made it possible to use more complicated network topologies.

```
query local1() {
  result ipsetup*.(ipaddress,services*.(port,transport_prot));
  assertion  ipsetup.ipaddress:InIpRange(128.111.48.*);
}

query local2() {
  result node*.(hostname,interfaces*.(mac,ipsetups*.(ipaddress,services*.(port,transport_prot))));
  assertion  ipsetup.ipaddress:InIpRange(128.111.48.*);
}

query department1() {
  result node*.(hostname,interfaces*.(mac,ipsetups*.(ipaddress,services*.(port,transport_prot))));
  assertion  ipsetup.ipaddress:InIpRange(128.111.46-49.*);
}

query department2() {
  result node*.(interfaces*.(ipsetups*.(netmask,ipaddress),link.network.netnumber));
  assertion  ipsetup.ipaddress:InIpRange(128.111.46-49.*);
}
```

Figure 6. Test queries used in the real network tests, expressed in NTL.

When using NetMap on the UCSB networks, the four test queries shown in Figure 6 were used. Two queries were run on the local class C network in the Reliable Software Lab (RSL), and two queries were run on four subnets in the Computer Science Department. The RSL network is connected by a switch, and the other subnets used in the tests have a similar topology. A router connects the different subnets. For the performance test 26 hosts in the RSL were used, and 22 hosts were used for the functionality tests.

6.1. Performance Test

The performance test focuses on how much time NetMap requires for a given task and how much overhead NetMap introduces. The test case is the "local1" query in Figure 6, which is a query of all the open ports in the RSL. In Figure 7, we compare the time required for these different methods. The first run is performed by using a shell script to perform a ping scan followed by a sequential port scan. The other two runs are performed by NetMap. In the NetMap sequential run, the port scan is also performed sequentially for all the input values. While in the parallel run, the number of execution threads for port scan was set to 10.

The NetMap sequential run and the shell script run take approximately the same time, which indicates that NetMap imposes very little overhead on the processing. The timing break down is discussed further in Section 6.2. The parallel run reduces the total time from about four hours to about half an hour, which is a factor of eight. All three runs find all the hosts in the network. The numbers of open ports, however, are slightly different. This is because a port may be opened or closed during the different test runs.

The reason that the scan took such a long time is that most hosts in the RSL are running local firewalls, which usually takes about 20 minutes per host to scan, while computers without a firewall usually can be scanned within 10 seconds. The fact that most of the port scan time is waiting for I/O is crucial for the parallel run. The data shows that the CPU usage is under five percent, even in the case of ten parallel port scans. For this reason, 15 threads were used for port and OS scans in the functionality tests.

6.2. Functionality Test

The functionality test cases are shown in Figure 6. The first test is a query for all the open ports in the RSL. The second query is for OS name, hostname, mac address, and all open ports on the hosts in the RSL. The third query asks for the same data from four subnets. The last query is for IP address, netmask, and network from the same subnets.

The tools used in the test were:

Ping Finds hosts that are up by issuing an ICMP echo message and listening for an ICMP echo-reply. Implemented using Nmap in ping scan mode.

NetARP Returns the ARP cache of a host given its IP address.

Nslookup Does a reverse DNS lookup on an IP address.

Osdetect Performs OS fingerprinting by sending various packets to the host and matching the result against a database of OS's TCP/IP profiles. Implemented as Nmap in OS detect mode.

Portscan Tries to connect to a range of ports on a given IP address. Implemented as Nmap in port scan mode.

ICMP_netmask Finds the netmask of an ipsetup by sending an ICMP netmask request to the IP address.

Netfind Takes the IP address and netmask of an `ipsetup` as input and returns a network IP address. The network IP address is the IP address ANDed with the net-

Testname	# of Host	# of open ports	Time
Shell Script	26	167	241:29
NetMap Sequential	26	162	250:59
NetMap Parallel	26	174	34:33

Figure 7. Performance testing of local network. Times are expressed in minutes and seconds.

mask. This tool does not do any active discovery. Implemented as a shell script.

Figure 8 contains the tool schedule chosen for each query, and the running time for each tool. The total column in the table shows the time it took to run the whole query. The processing time is the total time minus the sum of the tool times. This is the time NetMap uses to normalize the data and insert it into the NetDB.

In both of the local tests all the collected data was correct. NetMap was also able to discover most of the attributes queried. There were some problems detecting the OSs of some of the hosts (i.e., 7 out of 22 hosts did not get an OS mapping). The reason for this problem is that all the Linux boxes in the RSL run local firewalls. This prevents the OS discovery tool from fingerprinting the hosts.

In the first department scan, a higher percentage of the OSs were fingerprinted successfully compared to the local scan (7 out of 78 did not get a mapping). These were the same hosts as in the local test.

The second department scan was performed to determine if NetMap is able to group the scanned hosts into subnets. 46 out of 77 hosts got their netmask attribute detected and were successfully assigned to the correct network. The hosts that failed the netmask detection were not assigned to any network. However, these 31 hosts can be correctly assigned to the network by using the longest prefix match with known network addresses. The second department scan was not performed the same day as the first one, which explains the difference in the number of IP addresses.

By comparing the run times for the local and the first department scan one finds that the ping, NetARP, and nslookup run times increase approximately linearly with the number of hosts. The OS detect and port scan times do not increase much at all, while NetMap processing time increases considerably.

7. Conclusions and Future Work

This paper described the NetMap approach and the characteristics of the implementation of the first prototype. An initial network model has been designed by analyzing existing models used by network management, discovery, and analysis tools. A database-centered application, called NetDB, has been implemented to store an inventory of network objects conforming to the model, and two GUIs for browsing the database have been developed.

The database is populated by using composable network tools. The Network Tool Language has been defined to describe the tools in an abstract way. A language to describe network discovery tasks, called NetScript has also been defined. A prototype Query Processor component has been implemented. The Query Processor takes a NetScript task specification as input and produces a schedule of tool executions that will produce the desired results. It then executes each of the tools in the schedule and stores the result into the NetDB. In addition, a preliminary set of algorithms to deal with the reduction of inconsistent and/or redundant information has been designed and implemented. Tests have been performed to show that the implementation is capable of mapping network topology information, discover service configurations, and perform security analysis. The tests also showed that inconsistencies can be resolved. In order to perform the tests, a number of tools were integrated into NetMap. The amount of work needed to do this was minimal, which supports the claim that NetMap can be easily extended. Given more tools, it should be possible to map every feature of the network that is interesting from a security point of view.

Future work will focus on extending the current set of tool descriptions, improving the reduction algorithms, and using NetMap as the basis for intrusion detection. To be more specific, we plan to validate the flexibility of the Network Tool Language by describing a wide range of tools. By doing this the expressive power of the language as well as the overall integration power of the approach will be thoroughly tested. We also plan to perform additional analysis on the reduction algorithms that have been developed to deal with inconsistent and duplicated information.

Finally, NetMap will be used to support a new approach to detecting attacks, called the "status-based approach." The status-based approach identifies attacks by analyzing the differences between the intended network status as specified by the model and the actual network status as detected by the monitoring tools. This approach is similar to anomaly detection approaches. A status-based IDS does not rely on statistical models to represent the correct behavior of the system; therefore, it does not need to be "trained" over a long period of time. Furthermore, it can be used in highly

Testname	Ping	NetArp	Nslookup	Osdetect	Portscan	Total	Processing	# IPs
local1	:07	-	-	-	25:21	25:32	:04	22
local2	:06	:17	:07	26:42	25:20	52:40	:08	22
department1	:22	:47	:49	30:51	25:48	60:42	2:25	82

Testname	Ping	icmp_netmask	netfind	Total	Processing	# IPs
department2	:19	:34	:38	1:36	:05	77

Figure 8. Results of the real world tests. Times are expressed in minutes and seconds.

dynamic information systems where a well-defined pattern of usage cannot be determined.

Acknowledgments

This research was supported by the Army Research Office, under agreement DAAD19-01-1-0484 and by the Defense Advanced Research Projects Agency (DARPA) and Rome Laboratory, Air Force Materiel Command, USAF, under agreement number F30602-97-1-0207. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon.

The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Army Research Office, the Defense Advanced Research Projects Agency (DARPA), Rome Laboratory, or the U.S. Government.

References

[1] Big brother system and network monitor homepage. `http://bb4.com/`, 2002.

[2] J. Case, K. McCloghrie, M. Rose, and S. Waldbusser. Protocol operations for version 2 of the simple network management protocol (SNMPv2). Internet Engineering Task Force (IETF), RFC 1905, January 1996.

[3] D. M. T. Force. Common Information Model (CIM) Core Model. White Paper, August 2000. `http://www.dmtf.org`.

[4] Fyodor. Nmap – the network mapper. `http://www.insecure.org/nmap/`, 2002.

[5] O. Group. PROTOS Test-Suite: c06-snmpv1. `http://www.ee.oulu.fi/research/ouspg`, February 2002.

[6] Hewlett Packard. *Managing Your Network with HP OpenView Network Node Manager*, January 2000. Manufacturing Part Number: J1240-90035.

[7] Internet Security Systems. *Internet Scanner*, 2002. `http://www.iss.net/`.

[8] Nessus homepage. `http://nessus.org/`, 2002.

[9] M. Newnham. Getting Started with Tkined, January 1997. `http://wwwhome.cs.utwente.nl/~schoenw/scotty/docs/getstart.html`.

[10] J. Postel. Internet Control Message Protocol. RFC 792, 1981.

[11] J. Schonwalder and H. Langendorfer. Tcl Extensions for Network Management Applications. In *Proc. 3rd Tcl/Tk Workshop*, Toronto (Canada), July 1995.

[12] D. Wood, S. Coleman, and M. Schwartz. Fremont, A System for Discovering Network Characteristics and Problems. In *Proceddings of the USENIX Conference*, pages 335–348, January 1993.