NetSTAT: A Network-based Intrusion Detection Approach

Giovanni Vigna and Richard A. Kemmerer
Reliable Software Group
Department of Computer Science
University of California Santa Barbara
[vigna,kemm]@cs.ucsb.edu

Abstract

Network-based attacks have become common and sophisticated. For this reason, intrusion detection systems are now shifting their focus from the hosts and their operating systems to the network itself. Network-based intrusion detection is challenging because network auditing produces large amounts of data, and different events related to a single intrusion may be visible in different places on the network. This paper presents NetSTAT, a new approach to network intrusion detection. By using a formal model of both the network and the attacks, NetSTAT is able to determine which network events have to be monitored and where they can be monitored.

Keywords: *Security, Intrusion detection, Networks.*

1. Introduction

Network intrusions have become common and sophisticated. Attacking a system through a network provides the attacker with advantages that are not available when attacking a host. For example, network attacks often do not require any previous access to the attacked system and may be totally invisible from the audit trail produced by the attacked host. In addition, the use of firewalls to protect enterprise networks from the external Internet has often supported the design of open, efficient, and *insecure* internal networks. These networks are open to insider attacks. Although networks give the attacker additional advantages, networks, by their very nature, have some characteristics that intrusion detection systems (IDSs) can take advantage of. For instance, networks can provide detailed information about computer system activity, and they can provide this information regardless of the installed operating systems or the auditing modules available on the hosts. In addition, network auditing can be performed in a nonintrusive way, without notching the performance of either the monitored hosts or the network itself, and network audit stream generation cannot be turned off. Finally, network traffic has more precise and timely timing information than the audit records produced by the standard OS auditing facilities.

Network-oriented intrusion detection systems can be roughly divided into distributed IDSs and network-based IDSs. A survey of network-oriented IDSs is given in [7]. Distributed IDSs are an extension of the original, single-host intrusion detection approach to multiple hosts. Distributed IDSs perform intrusion detection analysis over audit streams collected from several sources, which allows one to identify attacks spanning several systems. Examples of this kind of systems are IDES [4] and ISOA [12]. Network-based IDSs take a different perspective and move their focus from the computational infrastructure (the hosts and their operating systems) to the communication infrastructure (the network and its protocols). These systems use the network as the source of security-relevant information. Examples of this kind of systems are NSM [2], DIDS [10], and EMERALD [9].

NetSTAT is a tool aimed at real-time network-based intrusion detection. The NetSTAT approach extends the state transition analysis technique (STAT) [3] to network-based intrusion detection in order to represent attack scenarios in a networked environment. However, unlike other network-based intrusion detection systems that monitor a single subnetwork for patterns representing malicious activity, NetSTAT is oriented towards the detection of attacks in complex networks composed of several subnetworks. In this setting, the messages that are produced during an intrusion attempt may be recognized as malicious only in particular subparts of the network, depending on the network topology and service configuration. As a consequence, intrusions cannot be detected by a single component, and a distributed approach is needed.

Network-level monitoring and distribution pose some new requirements on intrusion detection systems:

- Networks produce a large amount of data (events). Therefore, a network-based intrusion detection system (NIDS) should provide mechanisms that allow the Net-

work Security Officer (NSO) to customize event "collectors" so that they listen for only the relevant events.

- Relevant events are usually visible in only some parts of the network (especially in the case of large networks). Therefore, a NIDS should provide some means of determining where to look for events.

- A NIDS should generate a minimum amount of traffic over the network. Therefore, there should be some local processing of event data.

- A NIDS needs to be scalable. At a minimum, "local" NIDS should interoperate with other NIDSs (possibly in a hierarchical structure).

- For maximum effectiveness, NIDSs should be able to interoperate with host-based IDSs so that misuse patterns include both network events and operating system events.

The NetSTAT tool presented in this paper addresses the aforementioned issues. The NetSTAT approach models network attacks as state transition diagrams, where states and transitions are characterized in a networked environment. The network environment itself is described by using a formal model based on hypergraphs, which are graphs where edges can connect more than two nodes [1]. By using a formal representation of both the intrusions and the network, NetSTAT is able to address the first three issues listed above. The analysis of the attack scenarios and the network formal descriptions determines which events have to be monitored to detect an intrusion and where the monitors need to be placed. In addition, by characterizing in a formal way both the *configuration* and the *state* of a network it is possible to provide the components responsible for intrusion detection with all the information they need to perform their task autonomously with minimal interaction and traffic overhead. This can be achieved because network-based state transition diagrams contain references to the network topology and service configuration. Thus, it is possible to extract from a central database only the information that is needed for the detection of the particular modeled intrusions. Moreover, attack scenarios use assertions to characterize the state of the network. Thus, it is possible to automatically determine the data to be collected to support intrusion analysis and to instruct the detection components to look only for the events that are involved in run-time attack detection. This solution allows for a lightweight, scalable implementation of the probes and focused filtering of the network event stream, delivering more reliable, efficient, and autonomous components.

In order to address the remaining issues, namely scalability and integration with other IDS, NetSTAT is designed to be *interoperable*. A NetSTAT instance protecting a network can interact with other NetSTAT instances in an integrated way and it can interact with other IDS. Although the interoperability of NetSTAT is an important and interesting issue, due to space limitations it will not be discussed in this paper.

The remainder of this paper is structured as follows. Section 2 presents the NetSTAT architecture. Section 3 describes NetSTAT at work on an example attack scenario. Finally, Section 4 draws some conclusions and outlines future work.

2. NetSTAT Architecture

NetSTAT is a distributed application composed of the following components: the network fact base, the state transition scenario database, a collection of general purpose probes, and the analyzer. A high level view of the NetSTAT architecture is given in Figure 1. In the following subsections the design of the main NetSTAT components is presented.

2.1. Network Fact Base

The network fact base component stores and manages the security relevant information about a network. The fact base is a stand-alone application that is used by the Network Security Officer to construct, insert, and browse the data about the network being protected. It contains information about the network topology and the network services provided.

The network topology is a description of the constituent components of the network and how they are connected. The network model underlying the NetSTAT tool uses *interfaces*, *hosts*, and *links* as primitive elements. A network is represented as a hypergraph on the set of interfaces [11]. In this model, interfaces are nodes while hosts and links are edges; that is, hosts and links are modeled as sets of interfaces. This is an original approach that has a number of advantages. Because the model is formal, it provides a well-defined semantics and supports reasoning and automation. Another advantage is that this formalization allows one to model network links based on a shared bus (e.g., Ethernet) in a natural way, by representing the shared bus as a set containing all the interfaces that can access the communication bus. In this way, it is possible to precisely model the concept of network traffic eavesdropping, which is the basis for a number of network-related attacks. In addition, topological properties can be described in an expressive way since hosts and links are treated uniformly as edges of the hypergraph.

The network model is not limited to the description of the connection of elements. Each element of the model has

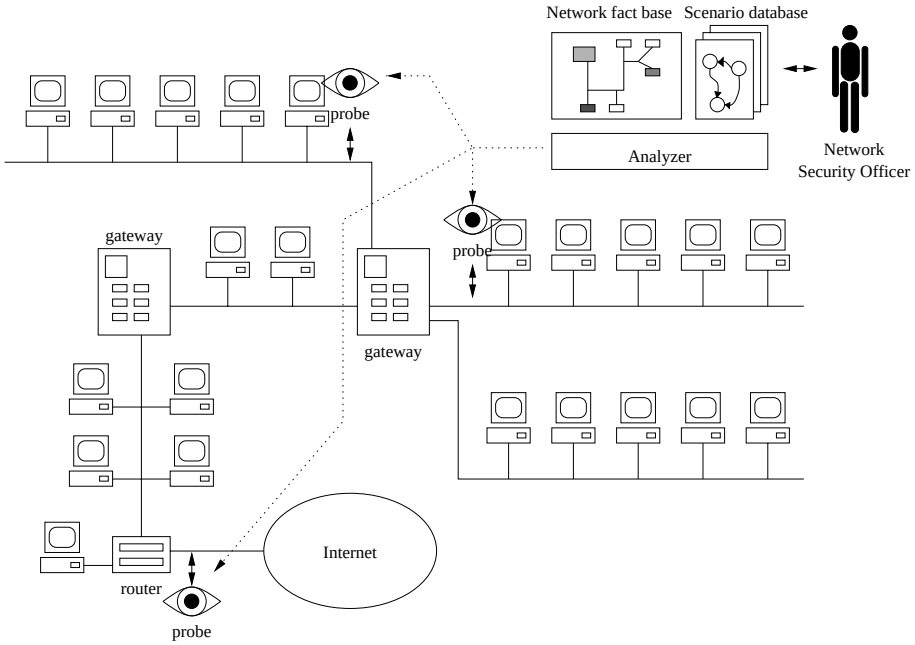

Figure 1. NetSTAT architecture.

some associated information. For example, hosts have several attributes that characterize the type of hardware and operating system software installed. The reader should note that in this model "host" is a rather general concept. More specifically, a host is a device that has one or more network interfaces that can be the (explicit) source and/or destination of network traffic. For example, by this definition, gateways and printers are considered to be hosts. Links are characterized by their type (e.g., Ethernet). Interfaces are characterized by their type and by their corresponding link- and network-level addresses. This information is represented in the model by means of functions that associate the network elements with the related information.

The network services portion of the network fact base contains a description of the services provided by the hosts of a network. Examples of these services are the Network File System (NFS), the Network Information System (NIS), TELNET, FTP, "r" services, etc. The fact base 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 network fact base contains information about how services are deployed, that is how services are instantiated and accessed over the network.

Figure 2 shows an example network. In the hypergraph describing the network, interfaces are represented as black dots, hosts are represented as circles around the corresponding interfaces, and links are represented as lines connecting the interfaces. The sample network is composed of five links, namely L_1, L_2, L_3, L_4, and L_5, and twelve hosts. Hereinafter, it is assumed that each interface has a single associated IP address, for example interface i_7 is associated with IP address a_7. The outside network is modeled as a *composite host* (the double circle in the figure) containing all the interfaces and corresponding addresses not in use elsewhere in the modeled network. As far as services are concerned, host `fellini` is an NFS server exporting file systems `/home` and `/fs` to `kubrick` and `wood`. In addition, `fellini` is a TELNET server for everybody. Host `jackson` exports an *rlogin* service to hosts `carpenter` and `lang`.

The Network Security Officer can access the network fact base through a graphic interface. The interface provides functions to design the network topology and to define the service infrastructure, as well as to browse the network-related information. Note that while the model is a complex, formal system, the interface used to manage the network fact base is intuitive and user-friendly.

NetSTAT is intended to represent large networks containing a large number of hosts that provide diverse services. For this reason, as an alternative to the graphic interface, all the information can also be inserted and retrieved by using ASCII-based tools. As a result, scripting languages like Perl or the Bourne Shell can be used to automate the retrieval of information from the network hosts, by querying the network information services (e.g., yellow pages, DNS) or by examining the configuration files of the involved hosts (e.g., `inetd.conf` on UNIX sys-

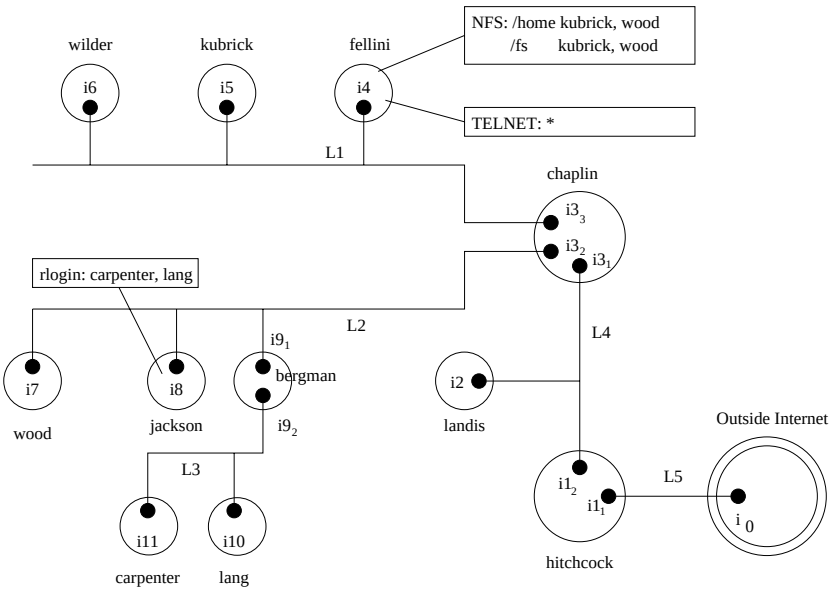

Figure 2. An example network.

tems). Large networks by their very nature are also subject to changes. Thus, the network fact base component will provide procedures that allow the Network Security Officer to verify the internal representation of the network against the actually deployed infrastructure in order to identify inconsistencies and incomplete or outdated information.

2.2. State Transition Scenario Database

The state transition scenario database is the component that manages the set of state transition representations of the intrusion scenarios to be detected. The state transition scenario database can be executed as a stand-alone application that allows the Network Security Officer to browse and edit state transition diagrams using a friendly graphic interface.

The state transition analysis technique was originally developed to model host-based intrusions [3]. It describes computer penetrations as sequences of actions that an attacker performs to compromise the security of a computer system. Attacks are (graphically) described by using *state transition diagrams*. *States* represent snapshots of a system's volatile, semi-permanent, and permanent memory locations. A description of an attack has a "safe" starting state, zero or more intermediate states, and (at least) one "compromised" ending state. States are characterized by means of *assertions*, which are functions with zero or more arguments returning boolean values. Typically, these assertions describe some aspects of the security state of the system, such as file ownership, user identification, or user authorization. *Transitions* between states are indicated by *signature actions* that represent the actions that, if omitted from the execution of an attack scenario, would prevent the attack from completing successfully. Typical examples of host-based signature actions include reading, writing, and executing files. For a complete description of the state transition analysis technique see [8]. For NetSTAT the original STAT technique has been applied to computer networks, and the concepts of state, assertions, and signature actions have been characterized in a networked environment.

States and Assertions

In network-based state transition analysis the state includes the currently active connections (for connection oriented services), the state of interactions (for connectionless services), and the values of the network tables (e.g., routing tables, DNS mappings, ARP caches, etc). For instance, both an open connection and a mounted file system are part of the state of the network. A pending DNS request that has not yet been answered is also part of the state, such as the mapping between IP address `128.111.12.13` and the name `hitchcock`. For the application of state transition analysis to networks the original state transition analysis concept of assertion has been extended to include both *static assertions* and *dynamic assertions*.

Static assertions are assertions on a network that can be verified by examining the network fact base; that is, by examining its topology and the current service configuration. For example, the following assertion:

```
service s in server.services|
  s.name == "www" and
  s.application.name == "CERN httpd";
```

identifies a service `s` in the set of services provided by host `server` such that the name of the service is `www` and the application providing the service is the CERN http daemon[1]. As another example, the following assertion:

```
Interface i in gateway.interfaces|
  i.link.type == "Ethernet";
```

denotes an interface of a host, say `gateway`, that is connected to an Ethernet link.

These assertions are used to customize state transition representations for particular scenarios (e.g., a particular server and its clients). In practice, they are used to determine the amount of knowledge about the network fact base that each probe must be provided with during configuration procedures.

Dynamic assertions can be verified only by examining the current state of the network. One examples is `NFSMounted(filesys, server, client)`, which returns true if the specified file system exported by `server` is currently mounted by `client`. Another example is `ConnectionEstablished(addr1, port1, addr2, port2)`, which returns true if there is an established virtual circuit between the specified addresses and ports. These assertions are used to determine what relevant network state events should be monitored by a network probe.

Transitions and Signature Actions

In NetSTAT, signature actions are expressed by leveraging off of an *event model*. In this model events are sequences of messages exchanged over a network.

The basic event is the *link-level message*, or *message* for short. A link-level message is a string of bits that appears on a network link at a specified time. The message is exchanged between two directly connected interfaces. For example the signature action:

```
Message m {i_x,i_y}|
  m.length > 512;
```

represents a link-level message exchanged between interfaces `i_x` and `i_y` whose size is greater than 512 bytes.

Basic events can be abstracted or composed to represent higher-level actions. For example, IP datagrams that are transported from one interface to another in an IP network are modeled as sequences of link-level messages that represent the intermediate steps in the delivery process. Note that the only directly observable events are link-level messages appearing on specific links. Therefore, the IP datagram "event" is observable by looking at the payload of one of the link-level messages used to deliver the datagram. For example, the signature action:

[1] The only (possibly) nonstandard notation used in the assertions is the use of "|" for "such that".

```
[IPDatagram d]{i_x,i_y}|
  d.options.sourceRoute == true;
```

represents an IP datagram that is delivered from interface `i_x` to interface `i_y` and that has the source route option enabled. This event can be observed by looking at the link-level messages used in datagram delivery along the path(s) from `i_x` to `i_y`. It is also possible to write signature actions that refer to specific link-level messages in the context of datagram delivery. For example, the signature action:

```
Message m in [IPDatagram d]{i_x,i_y}|
  m.dst != i_y;
```

represents a link-level message used during the delivery of an IP datagram such that the link-level destination address is not the final destination interface (i.e., the message is not the last one in the delivery process).

Events representing single UDP datagrams or TCP segments are represented by specifying encapsulation in an IP datagram. For example, the signature action:

```
[IPDatagram d [TCPSegment t]]{i_x,i_y}|
  d.dst == a_y and
  t.dst == 23;
```

denotes the sequence of messages used to deliver a TCP segment encapsulated into an IP datagram such that the destination IP address is `a_y` and the destination port is 23.

TCP virtual circuits are higher-level, composite events. A virtual circuit is identified by the tuple *(source IP address, destination IP address, source TCP port, destination TCP port)* and is composed of two sequences of TCP segments exchanged between two interfaces. Each of these two sequences defines a byte stream. The byte stream is obtained by assembling the payloads of the segments in the corresponding sequence, following the rules of the TCP protocol (e.g., sequencing, retransmission, etc.). The streams are denoted by `streamToClient` and `streamToServer`.

For example, the signature action:

```
TCPSegment t in
          [VirtualCircuit c]{i_x,i_y}|
  c.dstIP == a_y and
  c.dstPort == 80 and
  t.syn == true;
```

denotes a segment that has the SYN bit set and belongs to a virtual circuit established between interfaces `i_x` and `i_y` and that has destination IP address `a_y` and destination port 80.

Events at the application level can be either encapsulated in UDP datagrams or can be sent through TCP virtual circuits. In the former case, the application-level event can be referenced by indicating the corresponding datagram and specifying the encapsulation. For example, the signature action:

```
[IPDatagram d
     [UDPDatagram u
          [RPC r]]]{i_x,i_y}|
   d.dst == a_y and
   u.dst == 2049 and
   r.type == CALL and
   r.proc == MKDIR;
```

represents an RPC request encapsulated in a UDP datagram representing an NFS command.

In the TCP virtual circuit case, application-level events are extracted by parsing the stream of bytes exchanged over the virtual circuit. The type of application event determines the protocol used to interpret the stream. For example, the following signature action:

```
[c.streamToServer [HTTPRequest r]]|
  r.method == "GET";
```

is an HTTP GET request that is transmitted over a TCP virtual circuit (defined somewhere else as `c`), through the stream directed to the server side.

2.3. Probes

The probes are the active intrusion detection components. They monitor the network traffic in specific parts of the network, following the configuration they receive at startup from the analyzer, which is described in the following section. Probes are general purpose intrusion detection systems that can be configured remotely and dynamically following any changes in the modeled attacks or in the implemented security policy. Each probe has the structure shown in Figure 3.

The *filter* module is responsible for filtering the network message stream. Its main task is to select those messages that contribute to signature actions or dynamic assertions used in a state transition scenario from among the huge number of messages transmitted over a network link. The filter module may be configured remotely by the analyzer. Its configuration can also be updated at run-time to reflect new attack scenarios, or changes in the network configuration. The performance of the filter is of paramount importance, because it has strict real time constraints for the process of selecting the events that have to be delivered to the inference engine. In the current prototype the filter is implemented using the BSD Packet Filter [5] and a modified version of the *tcpdump* application [6].

The *inference engine* is the actual intrusion detecting system. This module is initialized by the analyzer with a set of state transition information representing attack scenarios (or parts thereof). These attack scenarios are codified in a structure called the inference engine table. At any point during the probe execution, this table consists of snapshots of penetration scenario instances (instantiations), which are not yet completed. Each entry contains information about the history of the instantiation, such as the address and services involved, the time of the attack, and so on. On the basis of the current active attacks, the event stream provided by the filter is interpreted looking for further evidence of an occurring attack. Evolution of the inference engine state is monitored by the *decision engine*, which is responsible for taking actions based on the outcomes of the inference engine analysis. Some possible actions include informing the Network Security Officer of successful or failed intrusion attempts, alerting the Network Security Officer during the first phases of particularly critical scenarios, suggesting possible actions that can preempt a state transition leading to a compromised state, or playing an active role in protecting the network (e.g., by injecting modified datagrams that reset network connections.)

Probes are autonomous intrusion detection components. If a single probe is able to detect all the steps involved in an attack then the probe does not need to interact with any other probe or with the analyzer. Interaction is needed whenever different parts of an intrusion can be detected only by probes monitoring different subparts of the network. In this case, it is the analyzer's task to decompose an intrusion scenario into subscenarios such that each can be detected by a single probe. The decision engine procedures associated with these scenarios are configured so that when part of a scenario is detected, an event is sent to the probes that are in charge of detecting the other parts of the overall attack. This simple form of forward chaining allows one to detect attacks that involve different (possibly distant) subnetworks.

2.4. Analyzer

The analyzer is a stand-alone application used by the Network Security Officer to analyze and instrument a network for the detection of a number of selected attacks. It takes as input the network fact base and a state transition scenario database and determines:

- which events have to be monitored; only the events that are relevant to the modeled intrusions must be detected;
- where the events need to be monitored;
- what information about the topology of the network is required to perform detection;
- what information must be maintained about the state of the network in order to be able to verify state assertions.

Thus, the analyzer component acts as a probe generator that customizes a number of general-purpose probes using

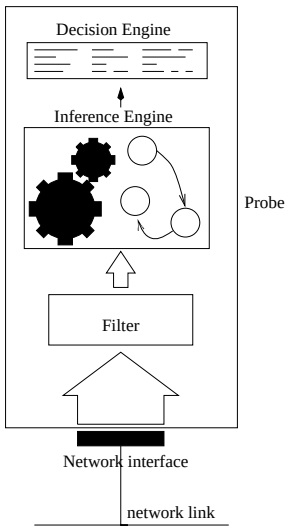

Figure 3. Probe architecture.

an automated process based on a formal description of the network to be protected and of the attacks to be detected. This information takes the form of a set of probe configurations. Each probe configuration specifies the positioning of a probe, the set of events to be monitored, and a description of the intrusions that the probe should detect. These intrusion scenarios are customized for the particular subnetwork the probe is monitoring, which focuses the scanning and reduces the overhead.

The analyzer is composed of several modules (see Figure 4). The network fact base and the state transition scenario database components are used as internal modules for the selection and presentation of a particular network and a selected set of state transition scenarios. The *analysis engine* uses the data contained in the network fact base and the state transition scenario database to customize the selected attacks for the particular network under exam. For example, if one scenario describes an attack that exploits the trust relationship between a server and a client, that scenario will be customized for every client/server pair that satisfies the specified trust relationship[2]. This customization allows one to instantiate an attack in a particular context. Using the description of the topology of that context it is then possible to identify what the sufficient conditions for detection are or if a particular attack simply cannot be detected given the current network configuration.

Once the attack scenarios contained in the state transition scenario database have been customized over the given network, another module, called the *configuration builder*, translates the results of the analysis engine to produce the configurations to be sent to the different probes. Each configuration contains a filter configuration, a set of state transition information, and the corresponding decision tables to customize the probe's decision engine.

[2] Thus, state assertions are treated as if they were universally quantified.

3. Example

The architecture of NetSTAT supports a well-defined configuration and deployment process. Firstly the Network Security Officer builds a database of attack scenarios. This database may include pre-modeled well-known scenarios or may be extended by the Network Security Officer following his/her perception of what an attack is and conforming to the local security policy. The attack scenario database is created and maintained using the state transition scenario database as a stand-alone application. Next, the Network Security Officer builds a network fact base describing the network to be protected. This task can be achieved by a combination of methods ranging from manual data introduction to automated information retrieval. This operation is performed by using the network fact base as a stand-alone application. In the next phase, the Network Security Officer invokes the analyzer. Using the integrated network fact base and state transition scenario database modules the Network Security Officer selects a particular network description and a set of scenarios that have to be detected. Again, this choice is dictated by the particular level of security that has to be achieved and by the network security policy. The Network Security Officer then starts the analysis and customization process. This process will be mostly automated, but in some circumstances it may require some interaction with the Network Security Officer, for example in order to manage situations where some attack scenarios cannot be detected. When the analysis is completed, the probe configurations will be created and sent to the probes installed

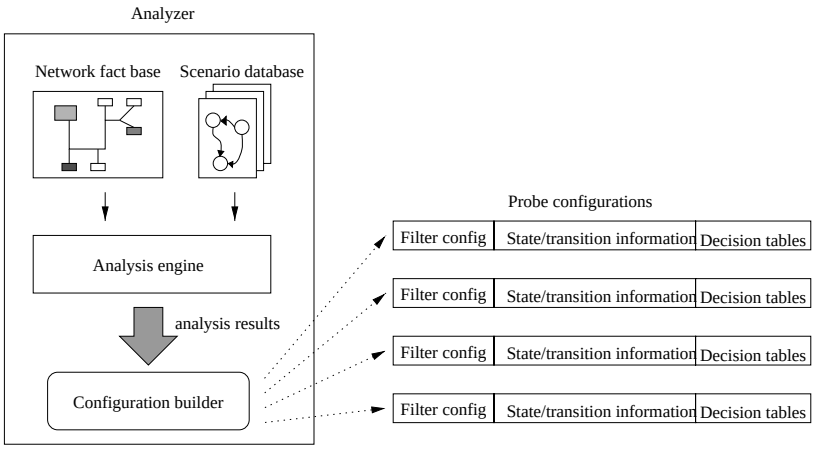

Figure 4. Analyzer architecture.

across the network.

The example attack considered is an active UDP spoofing attack. In this scenario an attacker tries to access a UDP-based service exported by a server by pretending to be one of its trusted clients, that is, by sending a forged UDP-over-IP datagram that contains the IP address of one of the authorized clients as the source address. The receiver of a spoofed datagram is usually not able to detect the attack. For this example, consider the network presented in Figure 2 and assume that host `lang` is attacking host `fellini` by providing an NFS request that pretends to come from `wood`, who is a trusted, authorized client. Host `fellini` receives the request encapsulated in a link-level message from `chaplin`'s interface i_{3_3} to `fellini`'s interface i_4. Host `fellini` has no means to distinguish this message from the final link-level message used to deliver a legitimate request coming from `wood`. Therefore, `fellini` cannot determine if the datagram is a spoofed one. The spoofing can be detected, however, by examining the message on link L_2. In this case, since the link-level message comes from `bergman`'s interface i_{9_1} while it should come from `wood`'s interface i_7, the datagram can be recognized as spoofed. In general, if one considers a single link-level message that encapsulates a UDP-over-IP datagram, the datagram may be considered spoofed if there is no path between the interface corresponding to the datagram source address and the link-level message source interface in the network obtained by removing the link-level message source interface from the corresponding link.

This attack scenario is described in Figure 5 using a state transition diagram. The scenario assumes that two networks have been defined, `Network` and `ProtectedNetwork`. `Network` is a reference to the network modeled in the fact base; `ProtectedNetwork` is a subnetwork that contains the hosts that must be protected against the attack.

The starting state (S_1) is characterized by assertions that define the hosts, interfaces, addresses, and services involved in the attack. The first assertion states that the attacked host `victim` belongs to the protected network. The second assertion states that there is a service `s` in the set of services provided by `victim` such that the transport protocol used is UDP, and service authentication is based on the IP address of the client. The third assertion states that `a_v` is one of the IP addresses where the service is available. The fourth assertion says that `a_t` is one of the addresses that the service considers as "trusted". The following assertions characterize the attacker. In particular, the fifth assertion states that there exists a host `attacker` that is different from `victim` and that doesn't have the trusted IP address. The sixth assertion states that `i` is one of the `attacker`'s interfaces.

The signature action is a spoofed service request. That is, a UDP datagram that pretends to come from one of the trusted addresses, although it did not originate from the corresponding interface. Actually the signature action is a link-level message `m` that belongs to the sequence of messages used to deliver an IP datagram from interface `i` to the interface associated with the address of the attacked host. The IP datagram enclosed in the message has source address `a_t` and destination address `a_v`. The IP datagram encloses a UDP datagram, whose destination port is the port used by service `s`. In addition, the message is such that, if one considers the network obtained by removing the message source interface from the corresponding link (i.e., `Network.detachFromLink(m.src)`), there is no path between the interface corresponding to the datagram IP source address and the link-level message source interface. For example, consider a link-level message exchanged between `bergman`'s interface i_{9_1} and `chaplin`'s interface i_{3_2}. The message is an intermediate step in the delivery of a UDP-over-IP datagram to `fellini`; the IP source address of the datagram is `wood`'s a_7. Intuitively, it

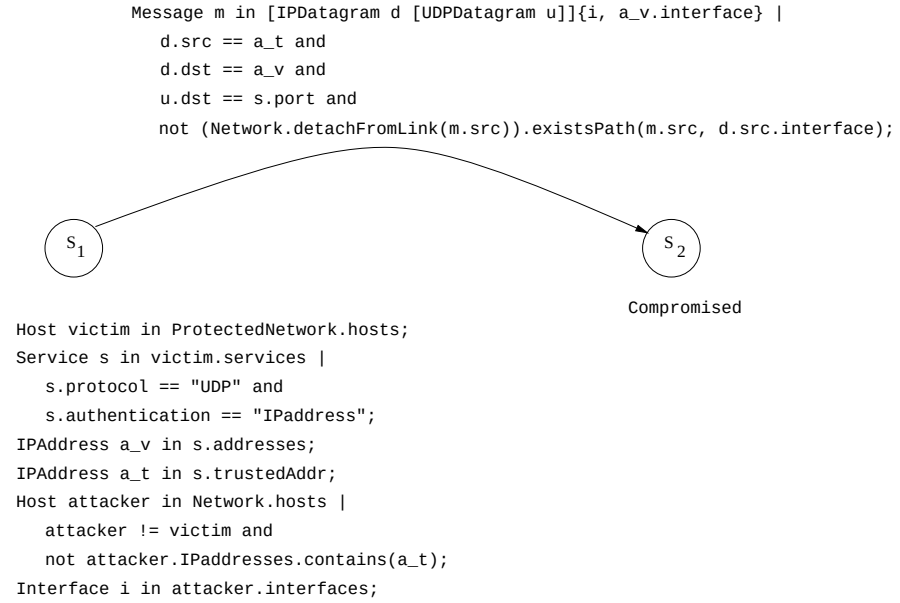

Figure 5. UDP spoofing attack scenario.

is clear that a message originated by `wood` and intended for `fellini` cannot come from one of `bergman`'s interfaces, because there is no path in the network that would require `bergman` to act as a forwarder of the datagram. One way to check for this is by removing the source interface of the message (i_{9_1}) and checking whether or not there still exists a path from the host whose IP address is the source of the datagram (`wood`) to the host that contains the interface that was removed (`bergman`). The second state (S_2) is a "compromised" state.

The analysis of the attack starts by identifying the possible scenarios in the context of a modeled network. Thus, the analysis engine determines all the possible combinations of victim host, attacked service, spoofed address, and attacker in a particular network. A subset of the scenarios for the network in Figure 2 is presented in Table 1. In all scenarios `fellini` is the attacked host, NFS is the service exploited, and the spoofed address can be `kubrick`'s or `wood`'s.

The next step in the analysis is to determine where the events associated with the signature action can be detected. For each of these scenarios, the analysis engine generates all the possible datagrams between the interface of the attacker and the interface of the victim. In practice, the engine finds all the paths between the interfaces defined by the scenario and, for each path, generates the sequence of messages that would be used to deliver a datagram. For each message the predicate contained in the clause of the signature action is applied. The messages that satisfy the predicate are candidates for being part of the detection of the scenario. For example, consider the scenario where `carpenter` is attacking `fellini` by pretending to be `wood`. In this case, the spoofed datagram is generated from interface i_{11} and delivered through three messages to `fellini`'s interface i_4. The first message is between `carpenter` and `bergman`, the second one is between `bergman` and `chaplin`, and the third one is between `chaplin` and `fellini`. Of these three messages only the first two satisfy the predicate of the signature action. Therefore, to detect this particular scenario one either needs a probe on L_3 looking for link-level messages from `carpenter`'s interface i_{11} to `bergman`'s interface i_{9_2}, or a probe on L_2 looking for messages from `bergman`'s interface i_{9_1} to `chaplin`'s interface i_{3_2}. In both cases, the IP source address is a_7, the destination IP address is a_4, and the destination UDP port is the one used by the NFS service. By analyzing all the scenarios, one finds that in order to detect all possible spoofing attacks it is necessary to set up probes on links L_1, L_2, and L_4.

4. Conclusions and Future Work

State transition analysis has proved to be an effective approach to host-based intrusion detection. This paper presents a further application of the state transition analysis approach, namely to detect network-based intrusions. The approach is based on formal models of attack scenarios (state transition diagrams) and of the network itself (network hypergraphs). These two models have been composed in order to determine the configuration and positioning of intrusion detection probes. The resulting architecture is distributed, autonomous, and highly customized towards the target network. In this way probes are more focused and the high volume event streams generated by networks can

victim	s	a_v	a_t	attacker	i
fellini	NFS	a_4	a_5	Outside	i_0
fellini	NFS	a_4	a_7	Outside	i_0
fellini	NFS	a_4	a_5	hitchcock	i_{1_1}
fellini	NFS	a_4	a_7	hitchcock	i_{1_1}
...	...	...	...	...	...
fellini	NFS	a_4	a_5	lang	i_{10}
fellini	NFS	a_4	a_7	lang	i_{10}
fellini	NFS	a_4	a_5	carpenter	i_{11}
fellini	NFS	a_4	a_7	carpenter	i_{11}

Table 1. Possible scenarios for the UDP spoofing attack.

be filtered resulting in decreased overhead.

The current prototype of the NetSTAT tool allows the Network Security Officer to define a network and the state transition diagrams describing the attacks. A number of sample networks have been constructed using the prototype network fact base component. In addition, network-based state transition representations for different flavors of UDP/TCP spoofing attacks, UDP race attacks, CGI-based attacks, remote buffer overflows, and denial of service attacks have been defined using the prototype state transition scenario database component. The analyzer is in a very initial stage of development. It provides a limited, *ad hoc* set of analytic capabilities, mostly related to topological analysis of network hypergraphs. Currently, the analyzer is not able to produce actual probe configurations and, although several configurations have been generated manually using the existing algorithms, it has not yet been tested on real networks.

Future work will focus on completion of the first prototype and on the refinement of its architecture. In addition, preparations are being made to test the prototype on a real network, containing several interconnected subnetworks, composed of heterogeneous hosts (PCs with Linux and Windows NT, SPARCstations with different versions of SunOS and Solaris, and IBM workstations with AIX).

Acknowledgments

This research was supported by the Defense Advanced Research Projects Agency of the Department of Defense under contract F30602-97-1-0207.

References

[1] C. Berge. *Hypergraphs*. North-Holland, 1989.

[2] L. T. Heberlein, G. Dias, K. Levitt, B. Mukherjee, J. Wood, and D. Wolber. A Network Security Monitor. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 296 – 304, Oakland, CA, May 1990.

[3] K. Iglun, R. A. Kemmerer, and P. A. Porras. State Transition Analysis: A Rule-Based Intrusion Detection System. *IEEE Transactions on Software Engineering*, 21(3), March 1995.

[4] H. S. Javitz and A. Valdes. The SRI IDES Statistical Anomaly Detector. In *Proceedings of the IEEE Symposium on Security and Privacy*, May 1991.

[5] S. McCanne and V. Jacobson. The BSD Packet Filter: A New Architecture for User-level Packet Capture. In *Proceedings of the 1993 Winter USENIX Conference*, San Diego, CA, January 1993.

[6] S. McCanne, C. Leres, and V. Jacobson. Tcpdump 3.4. Documentation, 1998.

[7] B. Mukherjee, L. T. Heberlein, and K. N. Levitt. Network Intrusion Detection. *IEEE Network*, pages 26–41, May/June 1994.

[8] P. Porras. STAT – A State Transition Analysis Tool for Intrusion Detection. Master's thesis, Computer Science Department, University of California, Santa Barbara, June 1992.

[9] P. Porras and P. Neumann. EMERALD: Event Monitoring Enabling Responses to Anomalous Live Disturbances. In *Proceedings of the 1997 National Information Systems Security Conference*, Baltimore, MD, October 1997.

[10] S. Snapp, J. Brentano, G. Dias, T. Goan, T. Heberlein, C. Ho, K. Levitt, B. Mukherjee, S. Smaha, T. Grance, D. Teal, and D. Mansur. DIDS (Distributed Intrusion Detection System) – motivation, architecture, and an early prototype. In *Proceedings of the 14th National Computer Security Conference*, Washington, DC, October 1991.

[11] G. Vigna. A Topological Characterization of TCP/IP Security. Technical Report TR-96.156, Politecnico di Milano, November 1996.

[12] J. Winkler. A Unix Prototype for Intrusion and Anomaly Detection in Secure Networks. In *Proceedings of the* 13^{th} *National Computer Security Conference*, Washington, D.C., October 1990.